68' MICRO JOURNAL

Australia A $ 4.75 New Zealand NZ $ 6.50
Singapore S $ 9.45 Hong Kong H $ 53.50
Malaysia M $ 9.45 Sweden 30 SEK

$2.95 USA

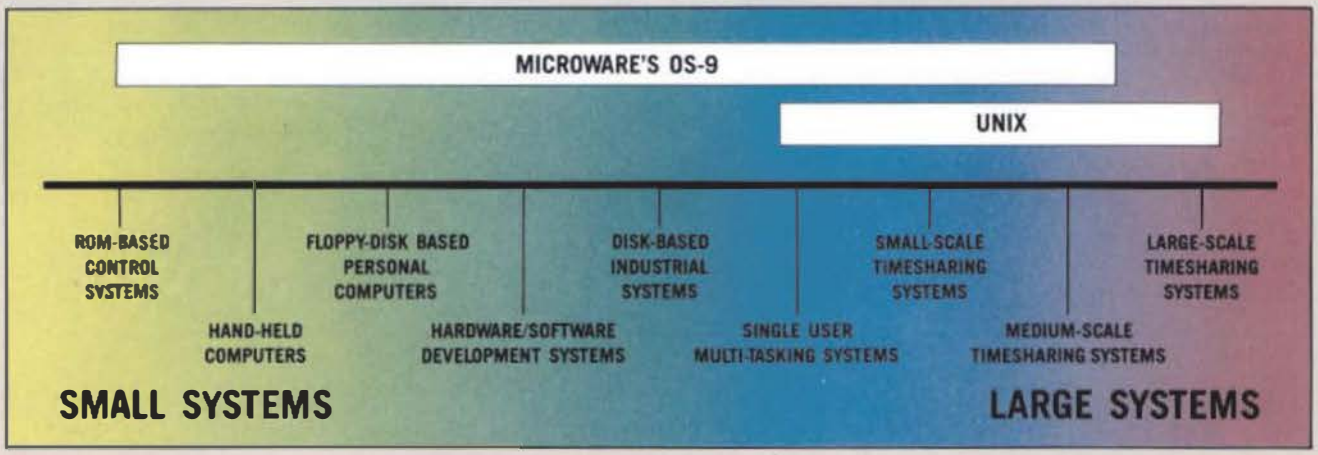
VOLUME VIII ISSUE III • Devoted to the 68XX User • March 1986

"Small Computers Doing Big Things"

PHOTO CREDIT: NASA

0 74470 12810 03

Only Microware's OS-9 Operating System Covers the Entire 68000 Spectrum



Is complicated software and expensive hardware keeping you back from Unix? Look into OS-9, the operating system from Microware that gives 68000 systems a Unix-style environment with much less overhead and complexity.

OS-9 is versatile, inexpensive, and delivers outstanding performance on any size system. The OS-9 executive is much smaller and far more efficient than Unix because it's written in fast, compact assembly language, making it ideal for critical real-time applications. OS-9 can run on a broad range of 8 to 32 bit systems based on the 68000 or 6809 family MPUs from ROM-based industrial controllers up to large multiuser systems.

OS-9'S OUTSTANDING C COMPILER IS YOUR BRIDGE TO UNIX

Microware's C compiler technology is another OS-9 advantage. The compiler produces extremely fast, compact, and ROMable code. You can easily develop and port system or application software back and forth to standard Unix systems. Cross-compiler versions for VAX and PDP-11 make coordinated Unix/OS-9 software development a pleasure.

Key OS-9 Features At A Glance

- Compact (16K) ROMable executive written in assembly language
- User "shell" and complete utility set written in C
- C-source code level compatibility with Unix
- Full Multitasking/multiuser capabilities
- Modular design - extremely easy to adapt, modify, or expand
- Unix-type tree structured file system
- Rugged "crash-proof" file structure with record locking
- Works well with floppy disk or ROM-based systems
- Uses hardware or software memory management
- High performance C, Pascal, Basic and Cobol compilers

SUPPORT FOR MODULAR SOFTWARE — AN OS-9 EXCLUSIVE

Comprehensive support for modular software puts OS-9 a generation ahead of other operating systems. It multiplies programmer productivity and memory efficiency. Application software can be built from individually testable softwar modules including standaid "library" modules. The modular structure lets you customize and reconfigure OS-9 for specific hardware easily and quickly.

A SYSTEM WITH A PROVEN TRACK RECORD

Once an underground classic, OS-9 is now a solid hit. Since 1980 OS-9 has been ported to over a hundred 6809 and 68000 systems under license to some of the biggest names in the business. OS-9 has been imbedded in numerous consumer, industrial, and OEM products, and is supported by many independent software suppliers.

microware® OS-9

AUTHORIZED MICROWARE DISTRIBUTORS

AUSTRALIA
MICROPROCESSOR CONSULTANTS
16 Bandera Avenue
Wagga Wagga 2650
NSW Australia
phone: 018-931-2331

ENGLAND
VIVAWAY LTD.
36-38 John Street
Luton, Bedfordshire
England LU1 2JE
phone: (0582) 423425
telex: 825115

JAPAN
MICROWARE JAPAN LTD.
3-8-9 Baraki, Ichikawa
Chiba 272-01, Japan
phone: 0473 (28) 4493
telex: 781-299-3122

SWEDEN
MICROMASTER SCANDINAVIAN AB
Sit Penggalan 7
Box 1309
S751-43 Uppsala
Sweden
phone: 018-138595
telex: 76129

SWITZERLAND
ELSOFT AG
Bankstrasse 9
5432 Neuenhof
Switzerland
phone: (41) 056-862724
telex: 57136

USA
MICROWARE SYSTEMS CORPORATION
1866 NW 114th Street
Des Moines, Iowa 50322
USA
phone: 515-224-1929
telex: 910-520-2535
FAX: 515-224-1352

WEST GERMANY
DR. KEIL GMBH
Perphystrasse 15
D-6905 Schriesheim
West Germany
phone: (0 62 03) 67 41
telex: 465025

ALL SYSTEMS INCLUDE:

- The CLASSY CHASSIS with a ferro-resonant, constant voltage power supply that provides + 8 volts at 30 Amps, + 16 volts at 5 Amps, and − 16 volts at 5 Amps.
- Gold plated bus connectors.
- Double density OMA floppy disk controllers.
- Complete hardware and software documentation.
- Necessary cables, filler plates.

YOU CAN EXPAND YOUR SYSTEM WITH:

MASS STORAGE

Dual 8" DSDD Floppies, Cabinet & Power Supply $1698.88
60MB Streamer (UniFLEX-020 only) $2400.00
1.6MB Dual Speed Floppy ..... (under development)

MEMORY

#67 Static RAM-64K NMOS (6809 Only) $349.67
#64 Static RAM-64K CMOS w/ battery (6809 Only) $398.64
#72 256K CMOS Static RAM w/ battery $998.72
#31 16 Socket PROM/ROM/RAM Board (6809 only) $268.31

INTELLIGENT I/O PROCESSOR BOARDS

significantly reduce systems overhead by handling routine I/O functions; freeing the host CPU for running user programs. This improves overall system performance and allows user terminals to be run at up to 19.2K baud. For use with GMX III and 020 systems.

#11 3 Port Serial-30 Pin (OS9) $498.11
#14 3 Port Serial-30 Pin (UniFLEX) $498.14
#12 Parallel-50 Pin (UniFLEX-020) $538.12
#13 4 Port Serial-50 Pin (OS9 & UniFLEX-020) $618.13

I/O BOARDS (6809 SYSTEMS ONLY)

#41 Serial, 1 Port $88.41
#43 Serial, 2 Port $128.43
#46 Serial, 8 Port (OS9/FLEX only) $318.46
#42 Parallel, 2 Port $88.42
#44 Parallel, 2 Port (Centronics pinout) $128.44
#45 Parallel, 8 Port (OS9/FLEX only) $198.45

CABLES FOR I/O BOARDS—SPECIFY BOARD

#95 Cable sets (1 needed per port) $24.95
#51 Cent. B.P. Cable for #12 & #44 ..... $34.51
#53 Cent. Cable Set $36.53

OTHER BOARDS

#66 Prototyping Board-50 Pin $56.66
#33 Prototyping Board-30 Pin $38.33
Windrush EPROM Programmer S30 (OS9/FLEX 6809 only) $545.00

CONTACT GIMIX FOR FURTHER DETAILS ON THESE AND OTHER BOARDS AND OPTIONS.

EXPORT MODELS: ADD $30 FOR 50Hz. POWER SUPPLIES.
ALL PRICES ARE F.O.B. CHICAGO.

GIMIX DOES NOT GUARANTEE PERFORMANCE OF ANY GIMIX SYSTEMS, BOARDS OR SOFTWARE WHEN USED WITH OTHER MANUFACTURERS PRODUCT.

GIMIX, Inc. reserves the right to change pricing, terms, and products specifications at any time without further notice.

GIMIX 2MHZ 6809 SYSTEMS

GMX 68020 SYSTEMS

Operating Systems Included	#49 OS9 GMX I/ and FLEX	#39 OS9 GMX II/ and FLEX	#79 OS9 GMX III/ and FLEX	#39 UniFLEX	#89 UniFLEX III	#020 UniFLEX VM
CPU included	#05	#05	GMX III	#05	GMX III	GMX 020 + MMU
Serial Ports Included	2	2	3 Intelligent	2	3 Intelligent	3 Intelligent
High Speed Static RAM	64KB	256KB	256KB	256KB	256KB	1 Megabyte
PRICES OF SYSTEMS WITH: Dual 80 Track DSDD Drives	$2998.49	$3398.39	$4898.79	N/A	N/A	N/A
25MB Hard Disk and one 80 track Floppy Disk	$5598.49	$5998.39	$7798.79	$5998.39	$8098.89	$13,680.20
72 MB Hard Disk and one 80 track	$7598.49	$7998.39	$9798.79	$7998.39	$10,098.89	$15,680.20
a 72MB + a 6MB removable pack hard disk and one 80 track floppy	$9098.49	$9498.30	N/A	$9498.39	N/A	N/A
a 72MB + a 12MB removable pack hard disk and one 80 track floppy	N/A	N/A	$11,298.79	N/A	$11,598.89	$17,180.20
GMX 6809 OS9/ FLEX SYSTEMS SOFTWARE OS9 + Editor, Assembler, Debugger	GMX I Included	GMX II Included	GMX III Included			
FLEX	Included	Included	Included			
GMXBUG Monitor	Included	Included	Included			
Basic 09, RunB (OS9)	Included	Included	Included			
RMS (OS9)	Included	Included	Included			
DO (OS9)	Included	Included	Included			
VDisk for FLEX	N/A	Included	Included			
RAMDisk for OS9	N/A	$125 option	Included			
O-FLEX	N/A	$250 option	Included			
Support ROM	N/A	N/A	Included			
Hardware CRC	N/A	N/A	Included			

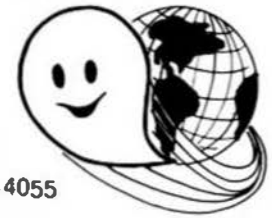
Available: Wide variety of languages and other software for use with either OS-9 or FLEX.

All GIMIX versions of OS9 can read and write RS color computer format OS9 disks, as well as the Microware/GIMIX standard format.

All OS9/FLEX systems allow you to software select either operating system.

TO ORDER BY MAIL: SEND CHECK OR MONEY ORDER OR USE YOUR VISA OR MASTER CHARGE. Please allow 3 weeks for personal checks to clear. U.S. orders add $5 handling if order is under $200.00. Foreign orders add $10 handling if order is under $200.00. Foreign orders over $200.00 will be shipped via Emery Air Freight COLLECT, and we will charge no handling. All orders must be prepaid in U.S. funds. Please note that foreign checks have been taking about 8 weeks for collection so we would advise wiring money, or checks drawn on a bank account in the U.S. Our bank is the Continental Illinois National Bank of Chicago, 231 S. LaSalle Street, Chicago, IL 60693, account number 73-32033.

GIMIX INC.
1337 WEST 37th PLACE
CHICAGO, ILLINOIS 60609
(312) 927-5510 • TWX 910-221-4055



'68' MICRO JOURNAL

Portions of the text for '68' Micro Journal were prepared using the following furnished Hard/Software:

COMPUTERS – HARDWARE
Southwest Technical Products
219 W. Rhapsody
San Antonio, TX 78216
S.9-5/8 DMF Disk-CDS1-8212W-Sprint 3 Printer

GIMIX Inc.
1337 West 37th Place
Chicago, IL 60609
Super Mainframe - OS9 - FLEX - Assorted Hardware

EDITORS – WORD PROCESSORS
Technical Systems Consultants, Inc.
111 Providence Road
Chapel Hill, NC 27514
FLEX - Editor - Text Processor

Stylo Software Inc.
PO Box 916
Idaho Falls, ID 83402
Stylograph - Mail Merge - Spell

Editorial Staff
Don Williams Sr. ..... Publisher
Larry E. Williams ..... Executive Editor
Tom E. Williams ..... Production Editor
Robert L. Nay ..... Technical Editor

Administrative Staff
Mary Robertson ..... Officer Manager
Joyce Williams ..... Subscriptions
Christine Lea ..... Accounting

Contributing Editors
Ron Anderson
Peter Dibble
Dr. Theo Elbert
Dr. E.M. Pass
Philip Lucido
Norm Commo
William E. Fisher
Carl Mann
Ron Voigts
Randy Lewis

Special Technical Projects
Clay Abrams K6AEP
Tom Hunl

Contents

Vol. VIII, Issue 3 March 1986

Send All Correspondence To:

Computer Publishing Center
68' Micro Journal
5900 Cassandra Smith Rd.
Hixson, TN 37343

Phone (615) 842-4600 or Telex 5106006630

68' Micro Journal is published 12 times a year by Computer Publishing Inc. Second Class Postage Paid ISSN 0194-5025 at Hixson, TN and additional entries. Postmaster: send form 3597 to 68' Micro Journal, POB 849 Hixson, TN 37343.

Subscription Rates

1 **Year $24.50 U.S.A.,** Canada & Mexico Add **$9.50** a Year. Other **Foreign Add $12 a Year** for **Surface, Airmail Add $48 a Year.** Must be in **U.S. currency!!**

Items or Articles For Publication

Articles submitted for publication should include authors name, address, telephone number and date. Articles should be **on either 5 or 8 inch disk** in STYLOGRAPH or TSC Editor format **with 3.5 inch column width.** All disks will be returned. Articles submitted on **paper** should be **4.5 inches in width** (including Source Listings) for proper reductions. **Please Use A Dark Ribbon!!** No Blue Ink!!! Single space on 8x11 bond or better grade paper. No hand written articles accepted. Disks should be in FLEX2 6800 or FLEX9 6809 any version or OS-9 any version.

The following TSC Text Processor commands **ONLY** should be used: **.sp** space, **.pp** paragraph, **.fi** fill and **.nf** no fill. Also **please do not format** within the text with **multiple spaces.** We will enter the rest at time of editing.

All STYLOGRAPH commands are acceptable except **.pg** page command. We print edited text files in continous text form.

Letters To The Editor

All letters to the editor should comply with the above requirements and **must be signed.** Letters of "gripes" as well as "praise" are solicited. We reserve the right to reject any submission for lack of "good taste" and we reserve the right to define "good taste".

Advertising Rates

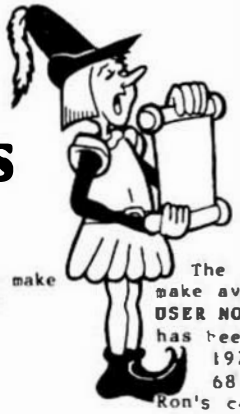
Commercial advertisers please contact 68' Micro Journal advertising department for current rate sheet and requirements.

Classified Advertising

All classified ads must be non-commercial. Minimum of **$9.50 for first 20 words** and .45 per word after 20. All classifieds must be **paid in advance.** No classified ads accepted over the phone.

THE 6800-6809 BOOKS

..HEAR YE.....HEAR



OS-9™ User Notes

By: Peter Dibble

The publishers of 68' Micro Journal are proud to make available the publication of Peter Dibbles **OS9 USER NOTES**

Information for the BEGINNER to the PRO, Regular or CoCo OS9

Using OS9
HELP, HINTS, PROBLEMS, REVIEWS, SUGGESTIONS, COMPLAINTS, OS9 STANDARDS, Generating a New Bootstrap, Building a new System Disk, OS9 Users Group, etc.

Program interfacing to OS9
DEVICE DESCRIPTORS, DIRECTORIES, "FORKS", PROTECTION, "SUSPEND STATE", "PIPES", "INPUT/OUTPUT SYSTEM", etc.

Programming Languages
Assembly Language Programs and Interfacing; Basic09, C, Pascal, and Cobol reviews, programs, and uses; etc.

Disks Include
No typing all the Source Listings in. Source Code and, where spplicable, assembled or compiled Operating **Programs.** The Source and the Discussions in the Columns can be used "as is", or as a "Starting Point" for developing **your OWN** more powerful Programs. Programs sometimes use multiple Languages such as a short Assembly Language Routine for reading a Directory, which is then "piped" to a Basic09 Routine for output formatting, etc.

BOOK $9.95

Typeset -- w/ Source Listings
(3-Hole Punched; 8 x 11)
Deluxe Binder - - - - - - - - - - $5.50

All Source Listings on Disk

1-8" SS, SD Disk - - - - $14.95
2-5" SS, DD Disks - - - $24.95

Shipping & Handling: $3.50 per **Book**, $2.50 per **Disk** set
* All Currency in U.S. Dollars
Foreign Orders Add $4.50 S/H (Surface)
or $7.00 S/H (Air Mail)
If paying by check - Please allow 4-6 weeks delivery

FLEX™ USER NOTES

By: Ronald Anderson

The publishers of 68 MICRO JOURNAL are proud to make available the publication of Ron Anderson's **FLEX USER NOTES**, in book form. This popular monthly column has been a regular feature in 68' MICRO JOURNAL SINCE 1979. It has earned the respect of thousands of 68 MICRO JOURNAL readers over the years. In fact, Ron's column has been described as the 'Bible' for 68XX users, by some of the world's leading microprocessor professionals. The most needed and popular 68XX book available. Over the years Ron's column has been one of the most popular in 68 MICRO JOURNAL. And of course 68 MICRO JOURNAL is the most popular 68XX magazine published.

Listed below are a few of the **TEXT** files included in the book and on diskette.

All TEXT files in the book are on the disks

LOGO C1	File load program to offset memory — ASM PIC
MEMOVE C1	Memory move program — ASM PIC
DUMP C1	Printer dump program — uses LOGO — ASM PIC
SUBTEST C1	Simulation of 6800 code to 6809, show differences — ASM
TERMEM C2	Modem input to disk (or other port input to disk) — ASM
M C2	Output a file to modem (or another port) — ASM
PRINT C3	Parallel (enhanced) printer driver — ASM
MODEM C2	TTL output to CRT and modem (or other port) — ASM
SCIPKG C1	Scientific math routines — PASCAL
U C4	Mini-monitor, disk resident, many useful functions — ASM
PRINT C4	Parallel printer driver, without PFLAG — ASM
SET C5	Set printer modes — ASM
SETBAS1 C5	Set printer modes — A-BASIC

NOTE: .C1,.C2, etc.=Chapter 1, Chapter 2, etc.

**Over 30 TEXT files included is ASM (assembler)-PASCAL-PIC (position independent code) TSC BASIC-C, etc.

Foreign Orders Add $4.50 S/H

Softcover — Large Format

Book only: **$7.95** + $2.50 S/H

With disk: 5" **$20.90** + $2.50 S/H

With disk: 8" **$22.90** + $2.50 S/H

Continually Updated In 68 Micro Journal Monthly

Computer Publishing Inc.
5900 Cassandra Smith Rd.
Hixson, TN 37343

(615) 842-4601
Telex 5106006630

"FLEX is a trademark of Technical Systems Consultants
"OS9 is a trademark of Microware and Motorola
"68' Micro Journal is a trademark of Computer Publishing Inc.

MUSTANG -020 Super SBC™

***

We Proudly Announce the MUSTANG-020 Super SBC*
"The one with the **REAL KICK!**"
Only from DATA-COMP

This is the NCC, world beater GMX SBC, in a super configuration. Data-Comp has mated it to a power plus power supply/stylish cabinet and your choice of floppy and/or hard disk drives. Available in several different configurations. (1) single board. (2) single board and regulators for your cabinet or mainframe and power supply. (3) single board - power supply and cabinet - your disk drives. (4) single board - power supply/cabinet - our drives configured to your specs, and ready to run. OS9 68K will be available as well as several other popular operating systems. Also all the popular OS9 68K software and Motorola 020-BUG will be available at a very reasonable price.

This system is the state-of-the-art star-ship. It runs rings around any other 68XXX SBC, and most mainframes. The speed and expanded RAM make this the "best buy" by a far stretch! A true multi-user, multi-tasking computer. So far advanced that even the experts don't call it a micro. Compared to the others, it isn't even a "horse race." And the price is certainly right. You can bet on this one!

So, will it be Turtle or **Thoroughbred?**

MUSTANG-020 System Prices
Effective November 1985

Mustang-020 SBC, wired & tested with 4 DB25 Serial ports pre-wired, ready to install with your cabinet, P/S, CRT and drives.....$2750.00

M020 Cabinet and P/S, for Mustang-020, less cables..... $269.95

M020 Cables, dual floppy or winchester, specify which - floppy or winchester.....$39.95

M020FC Floppy cabinet and P/S, holds and powers 2 thin-line floppies.....$79.95

M020F Floppy, 80 track, DD/DS.....$269.95

OS-9, SPECIAL Mustang-020 version.....$350.00

MC68881F/P co-processor.....$495.00

10 Megabyte Winchester.....$695.00

20 Megabyte Winchester.....$895.00

Winchester Controller (Xebec-).....$395.00

Note: for orders of complete systems (Mustang-020, cabinet & P/S, disk drives and OS-9, deduct 5% from total package. (limited time offer) See opposite page.

**** Special Winchester Notice ****

The Mustang-020 device descriptors will allow you to use practically **ANY** winchester drive supported by XEBEC controllers.

• • •

Include: $3.50 SBC, cables only S/H. Cabinets include $7.50 S/H. Complete System include $20.00. All checks must be in USA funds. Overseas specify shipping instructions and sufficient funds.

Dealer & Quantity Discounts Available

* Mustang-020 is trademark of Data-Comp-CPI

DATA-COMP
5900 Cassandra Smith Rd.
Hixson, TN 37343

VISA

SHIPPING
USA ADD 2%
FOREIGN ADD 5%
MIN. $2.50

☎ **(615)842-4600**
For Ordering
TELEX 5106006630

MUSTANG -020 Super SBC™

Mustang 020 Features

- 12.5 MHz MC68020 full 32-bit wide path Processor
  - 32-bit wide non-multiplexed data & address buses
  - On-chip instruction cache
  - Object-code compatible with earlier M68000 family processors (68000/68008/68010)
  - Enhanced instruction set - Coprocessor interface
- Optional 68881 Floating point Coprocessor (12.5 MHz)
  - Direct extension of 68020 instruction set
  - Full support of IEEE P754, draft 10.0
  - Transcendentals and other math functions
- 2 Megabytes of RAM (512K x 32-bit organization)
- Up to 256K bytes of EPROM (64K X 32-bits)
  - Uses four 2764, 27128, 27256, or 27512 EPROMs
- 4 Asynchronous serial I/O ports (2 x MC68681 DUART)
  - Software programmable baud rates to 19.2K
  - Standard RS-232 interface
  - Optional network interface on one port
- Buffered 8-bit Parallel I/O Port (1/2 MC68230)
  - Centronics-type parallel printer pinout
  - May also be used as parallel input port
- Expansion Connector for Additional I/O Devices
  - 16-bit data path
  - 256 byte address space
  - 2 interrupt inputs
  - Clock and Control Signals
- Time-of-Day Clock/Calendar w/battery backup
- Controller for up to Two 5 1/4" Floppy Disk Drives
  - Single or double sided
  - Single or double density
  - 48 or 96 tracks per inch (40/80 Track)
- Mounts Directly to a Standard 5 1/4" Disk Drive
- SASI Interface for Intelligent Hard Disk Controllers
- Programmable Periodic Interrupt Generator
  - For time-slicing and real-time applications
  - Interrupt rates from microseconds to seconds
  - Highly Accurate timebase (5 PPM)
- 5-bit sense switch, readable by the processor
- Hardware single-step capability

MUSTANG-020 Benchmarks **
Time Seconds

Type System	32 bit Int. Loop	Register Long Loop
IBM AT 7300 Xenix Sys 3	9.7	No Registers
AT&T 7300 UNIX PC 68010	7.2	4.3
DEC VAX 11/780 UNIX Berkley 4.2	3.6	3.2
DEC VAX 11/750 " " "	5.1	3.2
68008 OS9 68K 8 Mhz	18.0	9.0
68000 " " 10 Mhz	6.5	4.0
MUSTANG-020 68020 MC68881 OS9 16 Mhz	2.2	0.88
MUSTANG-020 68020 MC68881 UNIFLEX "	1.8	1.22

** loop:
```
Main()
{
  register long i;
  for (i=0; i < 999999; ++i);
}
```

Estimated MIPS - MUSTANG-020 - 2.5 MIPS
Motorola Specs: Burst up to 7 - 8 MIPS - 16 Mhz

☎ **(615)842-4600**
Telex 5106006630

For a limited time we will offer $400 Trade-In on your old Q--- 68008 or 68000 SBC, must be working properly and complete with all software and cables. Call for more information!

** ACTION PROVEN **

The MUSTANG-020 is already on the job! And winning aclaim in industry, commerce, business and several government agencies. The delivery times were close to schedule. We are hearing back nothing but praise (and more orders).

If you are considering the purchase of a Mustang-020, be advised that the price will increase in the second quarter of this year.

Experienced users are awed at the tremendous power and speed of the Mustang-020, from Data-Comp. Especially when compared with other 68XXX systems. Not only is it more practical than all the others, but it is much more cost efficient. Compare it to any other 68XXX and you will see why!

Dual 5" 80 trk. Floppy No Winchester			Winchester & 1 Floppy
020 Board	$2750.00		$2750.00
Cabinet	269.95		269.95
5"-80 trk Floppy(2)	539.90	(1)	269.95
Floppy cable	39.95		39.95
OS-9 68K	350.00		350.00
		Winchester cable	39.95
Total System	$3949.80	Winchester controller	395.00
Less 5%	-197.49	10 Megbyte Winchester	695.00
	$3752.31	Total System	$4809.80
S/H UPS	20.00	Less 5%	-240.49
Total	$3772.31		$4569.31
		S/H UPS	20.00
		Total	$4589.31
		With 20 Megbyte Winchester Add:	200.00
			$4789.31

NOTE: 68881 Co-Processor Add $495.00
UniFLEX $450.00
less $350.00 (OS-9) Add $100.00

Prices and Specifications subject to change.

Mustang-020 Software

OS-9
- OS-9.....$350.00
- Basic09.....300.00
- C Compiler.....400.00
- Fortran 77..... *400.00
- Pascal Compiler...400.00
- OPTASOFT-PASCAL..900.00
- Stylo-Graph.....495.00
- Stylo-Spell.....195.00
- Stylo-Merge.....175.00
- SCULPTOR+.....*Call
- CUM.....*Call

** See discount below

UniFLEX
- UniFLEX.....$450.00
- Screen Editor.....150.00
- Sort-Merge.....200.00
- BASIC/PreCompiler.300.00
- C Compiler.....350.00
- COBOL.....750.00
- Fortran 77.....450.00
- SCULPTOR+.....*Call

** See discount below

* New items

Standard system shipped 12.5 MHz
Add for 16 Mhz..68020....$400.00
Add for 16 Mhz..68881....$400.00
8 Port expansion board use two total of 20 RS232 ports wired & Tested.....each....$498.00

SCULPTOR+..We are USA distributors for SCULPTOR+. Call or write for site license or multiple discounts.

** Software Discounts

Call for software discounts from 10-70% for buyers of these systems, from Data-Comp. Limited offer. Call!

FLEX User Notes

Ronald W. Anderson
3540 Sturbridge Court
Ann Arbor, Mi 48105

PL/9

In the process of the PAT editor project, I ran across some interesting things regarding PL/9. PL/9 has a mode whereby you can let it compile a program and look at the code generated for each statement. I noticed some time ago that there seemed to be a lot of code generated where there was an .AND or .OR (logical and and or functions) in a statement. I decided to do some testing and see if some alternative structures would be more efficient. I defined some global BYTE variables A,B,C, and D. Then I let PL/9 compile:

```
IF A=B .AND C=D THEN
BEGIN
END;
```

I found that the byte count is 28 for that construct. The empty BEGIN END pair contribute no code to that count, which is what I wanted, since I wanted to compare the logical statement only. An equivalent function may be had using a compound IF statement. Moreover some speed gain may be realized with this structure.

```
IF A=B THEN IF C=D THEN
BEGIN
END;
```

If the A=B condition happens only once in a while and the C=D condition is common, this arrangement will cause the evaluation to stop when it is determined that A<>B. In addition, the code generated was only 16 bytes, a saving of 12.

```
WHILE A=B .AND C=D
BEGIN
END;
```

This structure generated 28 bytes also. I wondered if negating the second test and causing it to abort the loop would produce less code.

```
WHILE A=B
BEGIN
   IF C<>D THEN BREAK; /* FIRST STATEMENT OF LOOP */
    (LOOP STATEMENTS GO HERE)
END;
```

This alternative generated 21 bytes for a saving of 7 bytes. I had a thought for another alternative structure.

```
REPEAT UNTIL A=B .OR C=D;
```

This compiled 28 bytes again.

```
REPEAT
    (STATEMENTS GO HERE)
   IF C=D THEN BREAK; /* LAST STATEMENT OF LOOP */
UNTIL A=B;
```

This alternative generated only 17 bytes for a saving of 11. In all these cases the results are identical, though in the first case using compound IF THEN, the change cannot be made if there is an ELSE clause that relates to the whole logical .AND expression. If there is no ELSE clause, the substitution is direct.

So what? Who cares about a few bytes here and there? PAT had grown to 16.5K bytes, but by the time I had restructured all possible occurrences of the three above structures, I had saved over 1K of object code! Needless to say, an editor might have more than its share of logical .AND and .OR conditions, but I think there is a principle to be learned here. Depending on a compiler implementation, simple changes may produce fairly substantial code reduction. If you can think of an alternative structure it is worth a try. In one other case, I had a series of IF statements:

```
IF CH = 'T THEN ...
IF CH = 'B THEN...
.
.
```

I thought about changing it to a PL/9 CASE statement:

```
IF CH
   CASE 'T THEN...
   CASE 'B THEN...
.
.
.
END;
```

I was a little surprised to find that each case in the case statement added one byte to the code, over and above what was generated by the series of IF THEN statements. It would seem logical that getting the variable CH once and comparing it down a list of possibilities would take less code than getting it for each IF THEN statement in the first arrangement. Not so.

In most applications I would opt for the clarity of the .AND or .OR structure over the reduced code of the less clear construction. However, in the case of the editor, the smaller the program, the larger the edit buffer. Squeeze out 1K and I can have a larger edit buffer or several more features without further loss of the edit buffer.

HELP!

As I mentioned last time, I have been overwhelmed with work both fulltime and consulting. As a result some of your letters have been neglected for some time. I have gotten around to answering some of them, but several have gone unanswered. Many of the letters have been or will be answered in future columns.

Assembler Routines

I have some particularly good examples of optimization using assembler code, and I thought I would discuss them here. Frequently in a control or measurement application a need arises to multiply a 16 bit integer number by a small integer constant. For example, suppose I want to

be able to input a number as an ASCII string and convert it to 16 bit binary representation. The algorithm for doing it is simple,

1. Clear the variable NUMBER

2. Get the next character of the string (starting at the high order or leftmost one). If the character is not an ASCII digit, then goto 7.

3. Multiply NUMBER by 10.

4. AND the character with $0F to remove the $30 ASCII "bias".

5. Add the result of 4 to NUMBER

6. Go to step 2 and repeat

7. We're done. Result is in the variable NUMBER.

Most any measurement or control program would have an integer multiply subroutine, set up to multiply any two integer numbers and check for overflow. If you are in no hurry, you can just use that, but if you are optimizing code, you could consider this multiply as a special case that could be done a great deal faster than the standard multiply for several reasons. First you might well be able to assume that the input number couldn't overflow an integer variable. You could, for example, limit the input to four digits. You might be able to assume the number is always positive. Since the multiplier is small, (i.e. it is a "byte" value, less than 255) you can simplify the multiplication. You know the upper byte of the multiplier will be zero. You are multiplying by a positive 10, and you don't have to do a lot of manipulation to insure that the sign is properly handled. With all these limitations, the small integer multiply can be very short and fast. There are two ways to do it that come to mind immediately. You can do some add and shift manipulations or you can use the MUL instruction of the 6809.

I was curious to see which would be more efficient, so I coded the program both ways. Both assume the input is in a particular "buffer" called VALUE, and that the result is to be put in a double byte called TEMP.

First let's discuss the MULTEST program listing. Essentially what it does is to multiply the low order byte of the integer by ten, store the 16 bit result in TEMP, multiply the high order byte of the input number by 10 and add it to the high order byte. If you play computer a little, you will see that the other partial products will all be zero because the high order byte of the "10" is zero. If the number can't overflow there is no carry to worry about. This program with its test value of 533, runs 47 clock cycles down to but not including the RTS, since TSC Debug errors out on encountering an RTS without having previously found a JSR or BSR.

Though you might expect the MUL instruction to be the most efficient, now look at the ADTEST listing. This one uses the fact that a number is multiplied by two by shifting it left one place. The number is first loaded to ACCD and shifted left one place. Unfortunately the 6809 does not have an ASLD instruction. One must shift the low order byte (which moves the top bit to the carry flag) and then ROTATE the high order byte, which pulls the carry flag status into the low order bit. The instructions ASLB, ROLA accomplish multiplication by two. Two times the number is then pushed to the stack to save it for later, and the number is shifted left twice more, multiplying it by 8. Now we simply add 8N to 2N and the result is of course 10N. The result is stored in TEMP as the answer. This routine runs 38 clock cycles, and beats the one using the MUL instruction by 9 cycles.

Of course if the multiplier were 7 there would have to be a couple of saves and a couple of adds, and the MUL version would win. The point is that it pays to check alternative methods if you are really after speed. I would guess that most any 16 by 16 multiply routine that handles signs etc. would run on the order of 400 clock cycles. This optimization then is better by a factor of 8 or 10, than the blind use of the available multiply routine.

NOTE: I checked later and found that one of my compilers would do this multiply in 144 clock cycles but another one took 825.

The MUL version has the advantage that its execution time doesn't change appreciably if the multiplier is 7 or 15 or 31, whereas the shift and add version changes considerably. The shift and add version can be even faster for a multiply by 3 or 5, for example. The Shift version is two bytes smaller in terms of object code than the MUL version, a rather negligible difference. The MUL version could be modified to use a "byte" variable for the multiplier and it could stand as a small integer multiply routine. Either of these techniques could be extended to multiple byte integer values. Both are small enough to be used as macros so they appear in line where used, and avoid the JSR (BSR) - RTS execution times as well if speed were the all out objective. If I needed a fast string to binary routine, I would code the multiply by 10 in-line in the routine.

Now having said the above, let's put this in perspective a little bit. A routine to convert four digit ASCII strings to binary numbers would use the above MULTEN routine(s) four times. It would have some other overhead also. A fair estimate might be that we could convert a number in 250 clock cycles using one of the above routines. If the ASCII string is coming from operator input, we might as well not bother with the optimization, because even the slowest standard integer multiply would easily keep up with the operator. Suppose, however that the application is a data collector, and we have 1000 four digit numbers stored in memory to convert. (The discussion after this point will assume a 1 MHz 6809. All times may be divided by two for a 2 MHz system.) Our routine using the above could do it in 250 milliseconds. One using a standard multiply that takes 825 clock cycles, as I found one of my compilers to do, might do one four digit conversion in 3600 clock cycles. It would take 3.6 seconds to do 1000 conversions. In this case it would make sense to go to the trouble of the special routine. Suppose there were 10000 strings to convert? Then we're talking about 2.5 seconds vs 36 seconds. In that sort of application, there might be no question that the optimization is needed!

Good programming practice, obviously depends greatly on the application. A program is nearly always a trade-off between speed and size. If we optimize for speed only where we need it, and go for minimum size where speed is of little concern, we have done the very best we can do.

Many compilers allow the user to substitute Assembler code where it is necessary to improve execution speed. Generally, a higher level language is easier to read and maintain than assembler code. It is also easier to program in the higher level language. The general rule is therefore to write the program in the high level language and convert to assembler code only those routines whose execution speed limits the performance of the program.

I tried to do that with the new editor PAT. Pat is presently just at 15.5K of code. Of that about 3% or less is assembler. Yet the assembler code is probably being executed at least half of the time. When I dump the screen buffer to the edit buffer, for example, 2944 characters are moved to the edit buffer. The program handles lines of text on the high level, and calls an

assembler routine to move a line (128) characters from one place to another. The high level loop is executed 23 times, and the assembler loop 2944 times in dumping a screenful to the buffer. Save one clock cycle in the assembler loop and you have saved almost 3 milliseconds in a screen dump. Save one clock cycle in the higher level loop and you have saved 23 microseconds! It is obviously vastly more fruitful to optimize the character handling loop than the line handling loop.

In the case of a screen editor, what happens most? To insert a line, a number of other lines must be moved in the screen buffer. Same goes for deleting a line. To move to a different area of the file being edited, characters must be moved from the screen buffer to the edit buffer and vice versa. Right. Moving things around in memory is probably the most often h function in a line editor. Therefore, the most improvement in its performance will be made by optimizing the portions of the program that move characters around.

This discussion should lead to the formulation of a few rules for optimizing program execution time. First look for the one operation that the program does most. Where does it spend most of its time? If that place is a repeat loop of some sort, concentrate on removing every possible clock cycle from the instructions in that loop. Next, if the program is still not acceptably fast, find the next operation where the program spends a lot of time and reduce that time to a minimum. As a general rule, about 90% of the possible improvement in execution time may be made by optimizing two or three routines. Look for alternative ways to do the same thing. If you are clearing a big area of memory, just to give one simple example, you could do the following:

```
        CLRA
        LDX #BUFSTR  BUFFER START ADDRESS
LOOP    STA ,X+
        CMPX #BUFEND
        BNE LOOP
        RTS
```

If the buffer happens to contain an even number of locations, (or you can arrange it thus) however, you might do it considerably faster with:

```
        CLRA
        CLRB
        LDX #BUFSTR
LOOP    STD ,X++
        CMPX #BUFEND
        BLT LOOP
        RTS
```

In the second case, you would be clearing two locations at once each time through the loop. To clear the same amount of memory, would require half as many times through the loop. Yes, STD ,X++ does take longer than STA ,X+, but the CMPX and the Branch instruction are done half as many times when doing two bytes at a time. The savings are considerable.

Sometimes a little planning can simplify a loop and reduce its execution time. Suppose you could start your buffer that is going to be cleared, at address 0?

```
        CLRA
        CLRB
        LDX #BUFEND+1    PAST END OF BUFFER
LOOP    STD ,--X
        BNE LOOP
        RTS
```

By arranging the code so that the index register becomes zero as the exit condition there are now only 11 clock cycles used for each pass of the loop. STD ,--X uses 8 cycles and BNE uses 3. The first example above has a 13 clock cycle loop and the second a 14 cycle loop. Remember that the first method cleared one location at a time. Suppose we have a 1 mhz system and are to clear 1000 bytes of memory. The first method will take 13 milliseconds, the second will take 7, and the third 5.5. It should be clear that not only careful coding but also careful consideration of memory usage is important in minimizing execution time or optimizing a program. If there were no good reason NOT to use memory starting at $0000 for the buffer that is cleared by the example code above, obviously that would be the best choice. Perhaps there are other considerations however, that make that choice of memory a poor one. In that case the second method might be best. Someone once said "There ain't no free lunch". Programming and engineering in general usually work on the principle that you must give something up to get something else. In programming you give up size for speed. You give up speed for size. You optimize one part of a program at the expense of another part. Now I'll get down off my soapbox for this time.

More on Tandy

I must be fair. I recently sounded off about Tandy in this column. Well, like many big companies, they never did admit that they made a mistake, but after a lot of verbal kicking of the manager of the local Tandy Computer Center, he spent a lot of time on the phone and finally someone in the organization suggested that maybe the ROMs they sent us had been programmed incorrectly or were defective. We thought it strange that all three sets had the same defect, but they sent us three more sets via Federal Express. The defect obviously was that they had sent us version 2.5 the first time, since the new ones were version 2.9. I am happy to report that the new ROMs and the new MS-DOS finally cured the problems we reported ten months ago, without introducing new problems.

I can't say we're terribly excited about the great service or all the fumbling in the process, but it is good at last to be able to use the COM1 port as advertised, and to have the MODE utility operative. Now maybe with a little luck I can use Lattice "C" and write a communications program whereby we can pass text files between our FLEX systems and our Tandy MS-DOS systems.

- - -

```
 NAM ADTEST
 *
 * MULTIPLY 16 BIT VALUE BY 10 BY ADD AND SHIFT
 *
VALUE FDB 533
TEMP RMB 2
START LDD VALUE
 ASLB
 ROLA
 PSHS D
 ASLB
 ROLA
 ASLB
 ROLA
 ADDD ,S++
 STD TEMP
 RTS
 END START
```

```
 NAM MULTEST
 *
 * SPEED OF MUL FOR 16 BIT *10
 *
 ORG 0
VALUE FDB 533
TEMP RMB 2
START LDA #10
 LDB VALUE+1
 MUL
 STD TEMP
 LDA #10
 LDB VALUE
 MUL
 ADDD TEMP
 STD TEMP
 RTS
 END START
```

OS-9 User Notes

Peter Dibble
19 Fountain Street
Rochester, NY 14620

Computers and Rumors of Computers

Just yesterday I turned in the compiler I wrote for for a course this semester. It was one of biggest programs I have ever written. It represented many hours of work and several new bits of knowledge. One thing I learned had very little to do with compilers.

As the semester passed and the deadline approached I, and the rest of my class, worked ourselves into a panic. When your back is against the wall you'll try anything that might help. The straw I grasped at was a Sun workstation. (I wasn't the only one in my class to do that.) It did help. A lot.

A Sun is a 68010-based personal workstation (powerful person computer). Most of our Suns have a few megabytes of memory, a medium capacity hard disk, a large black and white screen, and an ethernet interface; some have no disk but extra memory.

When I first used a Sun I was impressed with the fine large letters on its exceptionally large screen. I didn't bother to learn its windowing software. I saw other people using windows and compared what they could do with what Mac users could do. The Sun windows didn't do well by that comparison so I didn't spend the effort to learn to use them.

I was wrong. I should have learned. It seemed a little strange that all my associates would crowd the Suns while I used the empty terminals. Did they really like working on small machines that much? Wasn't I the leading microcomputer fanatic in the department? I finally picked up the documentation and spent a half an hour playing with Sun windows. I was right, they are nowhere near as nice as what a Macintosh has. I was also wrong, they are plenty good enough.

I never moved all my work to a Sun. When I moved off my OS-9 system to get access to the tools at the department I moved to one of our VAX/750s. I stayed there and used a Sun mostly as a terminal. In the heat of the final rush I had things set up like this.

* One small window was running a shell on my Sun.
* A 24 by 80 and a 54 by 80 window were logged into the vax that I used for most of my work.
* A 24 by 80 window was logged into the department's main computer (for mail and such).
* There was space left over on the screen for a clock and a few other tricks.

The best thing about that layout was that I could look at a file for reference while editing another (without paging through a listing). When I was using the debugger or dealing with compiler errors I would watch errors on one window and edit in another. When I needed to hang onto something for a while I'd use the mouse (I forgot to mention: there's a mouse) to cut the text out of one window and paste it into an idle window.

It's hard to describe why, but having several BIG windows on the screen made my work much easier. I'd guess that I worked about thirty percent faster because of them. I'm not sure whether little Macintosh windows would have had the same effect. I could put four 24x80 windows on the Sun's screen at the same time with no window overlapping another and run different programs in each one. Sometimes I did just that.

There is windowing software for OS-9 that is about as good as the Sun stuff. Unfortunately, I don't believe any computer available in this country has them. I would guess that there are two reasons for this. Windowing requires pretty sophisticated display management. OS-9 doesn't know how to do that for a generalized terminal. The OS-9 windowing package has to be tailored for specific display hardware. Preferably a memory-mapped video device.

The windowing software also consumes a lot of memory. You have to add the windowing software to OS-9, then you have to have enough memory left to run a few concurrent processes -- or why bother with windows. I'd guess you'd need at least 128K for a 6809 and 256K for a 68K.

You can't buy a computer off the shelf in this country with those qualities. The CoCo is the only OS-9 computer with a standard memory-mapped display but it doesn't have the memory. Maybe the next CoCo will come with windows. By the way, I'm still hoping for a new CoCo around, say, March of next year (1986).

You've probably noticed by now that I'm an optimist. My predictions are based on lots of information but I tend to get excited and overly hopeful. For example, I expected a port of OS-9 to the AT&T Unix PC before now. It sounds like AT&T has decided to keep internal documentation for that machine secret even from the likes of Microware.

Since interviewing AT&T people about the Unix PC I've given it a lot of thought. It gets brought to my attention every day or so when a message shows up on the network asking how to do something with the Unix PC that should be in the documentation.

It interesting to speculate about the motivations for AT&T's secretiveness. From what I hear there's no sexy new technology in there for them to protect as a trade secret. That leaves three possible reasons for the missing documentation: they're ashamed of the machine, they don't understand it themselves, or they'll go to great lengths to control it. None of these are good for users. My conclusion is: Stay away from that computer.

My optimistic thoughts about the Atari ST and the Commodore Amiga seem to be turning out better. This is one of those things that I found out about after agreeing not to write too much about it yet. I can tell you some. A company is porting OS-9 to both the Atari

ST and the Amiga. The company has experience with OS-9/68K. They plan to offer a package with OS-9, Basic-09, C, and Pascal. Of course this is all in the planning stage. I get the impression that projecting marketing plans is about as hard as predicting software completion dates. Speaking of software completion, the ports are coming along nicely.

Interesting side issue: this same company (who shall remain nameless) MIGHT also do networking hardware for the new OS-9 network file system.

My guess is that they'll be lucky if they can get the price for the OS-9-plus-languages package down to six-hundred dollars. An ST with a monochrome monitor costs about eight-hundred dollars. People who buy inexpensive computers aren't going to flock to an expensive operating system no matter how good it is. I don't expect that we'll find ourselves buried with Atari or Amiga users, but this could be the source of a few tens of thousands of new OS-9 users. They would also be nice environments for windows!

Of course, if Atari or Commadore decided to pick up OS-9 things could be different. Then we could be talking about many new members for our little community. Both the ST and the Amiga are nice machines that deserve to suceed.

"C" User Notes

Edgar M. (Bud) Pass, Ph.D.
1454 Latta Lane
Conyers, Ga 30207

INTRODUCTION

This chapter begins a discussion of the design and implementation of a text editor written entirely in the C language. Rather than covering the details of one editor, as was done in an earlier chapter with ED, it discusses each function of a text editor and potential methods of design and implementation. In the process, many C concepts, such as recursive structures and basic file-handling, are covered to some degree.

TEXT EDITOR CONCEPTS

Essentially everyone who programs a computer uses a text editor. Many who use them but do not program them use a text editor. It may be as trivial as a line-oriented editor, such as those provided by many BASIC compilers, in which lines may be added, deleted, or changed, but only on an entire-line basis. It may be as sophisticated as a professional word-processor, with a very large number of commands and features, perhaps requiring a keyboard with many special-purpose buttons. The text editors most programmers use lie somewhere between these extremes. Many programmers are familiar with several text editors, because of convenience or dependencies between text editors and language processors.

Most text editors perform at least the following functions:

- internal processing
- file input
- file output
- keyboard input
- display output

Some specialized text editors, such as those used in control systems, do not support file input and output, but essentially all general-purpose text editors support these functions.

This discussion covers potential means of implementing each of these functions using the C language. Note that the functions are all interdependent. If the internal data structure is changed, the file input and output functions must be changed. Also note there is no best manner in which to design and implement a text editor.

It may be as trivial as a line-oriented editor, but essentially everyone who programs a computer uses a text editor.

The example program at the end of this chapter illustrates many of the design points covered in this discussion. It omits the expansion of the keyboard and display functions, as they are covered later, since they deserve so much attention. It illustrates only a simple set of internal processing functions.

INTERNAL PROCESSING

The internal processing functions of a text editor determine most of its characteristics. These include whether the editor is line or screen-oriented, it is "what you see is what you get" or not, it has a spartan or a rich command set, it has single or multiple

windows, etc. Methods of implementing some of these characteristics are discussed in later chapters.

One important design decision which must be made before most of the other design decisions is the internal data structure which will contain the working version of the file being edited.

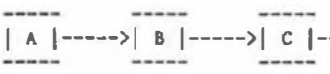
The designer could decide that the maximum allowable line length will be no wider than the display device and logically pre-allocate main memory in fixed one-line blocks. This provides a simple data structure in which the display window moves rigidly thru the working file, displaying exactly what is in that memory area. Changes within lines are made by modifying specific areas within the logical window storage in memory. Lines may be added or deleted only by copying all subsequent lines down one block or up one block. The n-th line in the working file may be located by a simple address computation.

A modification to the fixed-length data structure is the variable-length data structure. Rather than allocating a fixed block for each line, each line is represented by its text, followed by a delimiter, such as a carriage return, line feed, null, delete, or other single ASCII character. This eliminates the inefficient use of storage of the fixed-block data structure, while maintaining most of the simplicity of that structure. Unfortunately, adding and deleting lines are just as time-consuming, changing lines may also involve moving all subsequent contents of the working file, and locating the n-th line must be done by counting delimiters from the current window or from the beginning of the working file. Nevertheless, this data structure is used by a significant number of commercial text editors.

The data structure which is used in the text editor being designed here is the linked list. Depending upon the design parameters, the line length may be fixed or variable. For the sake of simplicity of exposition, a fixed length of 80 characters and a single forward pointer were chosen; however, a variable line length and single and backward pointers should be used in any real-world editor. The linked-list data structure solves the problems of adding, deleting, and changing lines, as lines are logically, not physically, sequentially structured. The single forward pointer makes traversing the working file in a forward direction faster than traversing it in a backward direction, but most editing is done in a forward direction, not in a backward direction.

The C code required to add and delete linked structure members is illustrated in the example program at the end of this chapter. Note the recursive definition of the linked-list structure. The collection and reuse of deleted lines is not included, to simplify the program. It could involve linking the deleted structure elements on a free list, then reusing these elements rather than allocating new storage whenever the free list is not empty. It could also be implemented by using the C library function "free" to return deleted lines for potential future memory allocation.

The following diagrams may assist in visualizing the process of adding and deleting items to and from a linked list. Assume that the list is originally structured as follows:



It would be structured as follows after the addition of an element D between A and B:

It would be structured as follows after the deletion of element B:

Note that none of the elements of the original list have been moved, and that the addition and deletion operations involved only the modification of pointers. The modification of the first and last pointers in the linked list normally requires special consideration. A simple avoidance of this problem is to add dummy first and last entries in the data structure which are never displayed to the user.

The processing of internal data structures is covered at greater length in later chapters.

FILE INPUT AND OUTPUT

Assuming that sequential files are used for input and output, the file-handling functions of a text editor are usually relatively simple. Many text editors support not only the initial reading and final writing of files, but the insertion of file contents into the working version of a file and the extraction of a new file from the working version of a file.

Some text editors support the editing of files larger than the amount of memory dedicated to the text buffer, although the user is normally aware of the situation, either because special commands are required to load the next block of text from the file, or because of the long time delay required to page the working file to and from auxiliary storage.

The file input and output functions must correctly process the internal data structure used by the text editor. For reasons discussed above, this is often a linked-list structure.

The file input routine must insert new text into existing text, if any, at the appropriate locations, perhaps requesting the allocation of additional memory either from a free-memory list, from a pre-allocated area, or from the system.

The file output routine is somewhat simpler than the file input routine, not normally modifying the internal data structure as it outputs the requested text, and thus traversing the relevant portion of the data structure to locate the text to be written.

KEYBOARD INPUT AND DISPLAY OUTPUT

The keyboard and display functions may be reasonably simple or may be extremely complex, depending upon the support provided by the operating system and the level of generality desired for the editor in terms of terminals and systems upon which it is designed to operate properly.

Operating systems such as UNIX provide a substantial amount of support for terminal independence. UNIX provides a package of subroutines called "terminfo" which allows the system administrator to symbolically define the characteristics of each type of terminal to be used on the system and allows the user to specify the particular type of terminal by code name. An application programmer utilizing the terminfo package is able to refer generically to the basic functions of a keyboard or a display device. UNIX provides an even higher-level package of subroutines called "curses" which provides further insulation from the details of terminal input and output and provides optimization of screen output and update.

If a text editor is to be system and terminal independent, the keyboard and display functions must be designed in as general a manner as possible. Usually,

this means that the user is responsible for defining a module to be linked with the editor to interface with a given terminal and system or that the user is responsible for constructing a table which the editor can read to determine the characteristics of a given terminal and system.

The display and keyboard functions are represented in the example program provided at the end of this chapter by "initscreen", "inkeyboard", and "outscreen". The "inkeyboard" function returns a code representing which program function key was struck. The pointer array to the data structure used by the display and keyboard functions is named "scrinx". It is part of the interface which allows data to be received from the keyboard and sent to the display to and from the internal processing functions. Details of such functions are covered in later chapters.

C PROBLEM

The program below extends the help program from an earlier chapter by implementing the standard UNIX-like wildcard meta-characters "?" and "*" in keyword strings. "?" matches any single character and "*" matches any string of zero or more characters.

The recursive method used to match the "*" meta-character significantly simplifies the string matching algorithm. Without recursion, a table is required to remember how far to backtrack in the comparison when a mismatch occurs. With recursion, the necessary bookkeeping is done automatically in the stack; when a mismatch occurs, the location to which to backtrack is in the stack.

```
/*
** help program      called as:     help helpfilename
**   help file has following format:
**       *keyword
**       text
**       :
**       text
**       *keyword
**       text
**       :
**       text
**       *     (FLEX users must end with dummy keyword)
*/

#include <stdio.h>

#define TABSIZE 256
#define TABENT  64

main(argc,argv)
int argc,**argv;
{
    FILE *input;
    int i = 0, j, k;
    char string[128], query[128];
    char keyword[TABENT + 1][TABSIZE + 1];
    long loc[TABSIZE + 1];

    if (argc < 2)
    {
        fputs("usage: help filename\n",stdout);
        exit(1);
    }
    if ((input = fopen(*++argv,"r")) == NULL)
    {
        fputs("Can't open ",stdout);
        fputs(*argv,stdout);
        fputs(".\n",stdout);
        exit(2);
    }
    setbuf(input,NULL);
  while (fgets(string,128,input))
      if ((*string == '*') && (*(string + 1) >= ' '))
          if (i >= TABSIZE)
              break;
          else
          {
              string[strlen(string) - 1] = 0x00;
              strncpy(keyword[i],string + 1,TABENT);
              loc[i++] = ftell(input);
          }
    rewind(input);

    while (1)
    {
        fputs("Enter keyword. ",stdout);
        if ((!fgets(query,128,stdin)) ||
            ((j = strlen(query) - 1) < 1))
            break;
        query[j] = 0x00;
        for (j = 0; j < i; ++j)
            if (wildcard(query,keyword[j]))
            {
                if (!fseek(input,loc[j],0))
                {
                    while (fgets(string,128,input)
                        && (*string != '*'))
                        fputs(string,stdout);
                }
            }
    }
    fclose(input);
    exit(0);
}
    wildcard(pattern, string)
    char *pattern, *string;
    {
        char c, d, r = 0, *p = pattern, *s = string;

        while (1)
        {
            switch (*p)
            {
            case '*':
                if ((!*s) || (!*++p))
                    goto exit1;
                while (*s)
                    if (wildcard(p, s++))
                        goto exit1;
                goto exit0;
            case '?':
                break;
            case '\0':
                r = !*s;
                goto exit0;
            default:
                if (*p != *s)
                    goto exit0;
            }
            ++p;
            if (*s)
                ++s;
        }
    exit1:
        r = 1;
    exit0:
        return (r);
    }
```

For the next C problem, code a function and a test driver which will compute the factorial of a given positive integer. The factorial of a number is the product of all positive integers less than or equal to the number. Code the function using both recursive and non-recursive techniques, and determine which technique provides the fastest and which provides the most compact function definition. Determine the largest positive integer for which the program will correctly compute the factorial exactly.

EXAMPLE C PROGRAM

Following is this month's example C program; it illustrates the file-handling and certain internal functions of a text editor.

```
    #include "driver.h"

    struct groups
    {
        struct groups *link;
        char line[80];
    };

main(argc,argv)
int argc;
char **argv;
{
    FILE *text;
    char *p, *q, *r, *s, *t, *u;
    char string[256], temp[20], filename[64];
    short int i = 0, j, k = 0, key, recs = 0;
    struct groups *first, *this, *that, *page, *last, *prev;
    struct groups *valid, *links[19];
```

```
    initscreen(*++argv);
    strcpy(filename,*(argv + 1));
    first = this = that = page = last = prev = valid = NULL;
    if ((text = fopen(filename,"r")) == NULL) /* try to open file */
        goto setup;
    while (fgets(string,255,text) != NULL)
    {
        if ((last = (struct groups *)
            calloc(1,sizeof (struct groups))) == NULL)
            goto termin;
        ++recs;
        strcpy(last->line,string);
        if (first == NULL)
            first = last;
        else
            this->link = last;
        (this = last)->link = NULL;
    }
    fclose(text);

setup:
    while ((first == NULL) || (recs % 18))
    {
        if ((last = (struct groups *)
            calloc(1,sizeof (struct groups))) == NULL)
            goto termin;
        ++recs;
        last->line[0] = 0x00;
        if (first == NULL)
            first = last;
        else
            this->link = last;
        (this = last)->link = NULL;
    }
    for (i = 1; i < 19; ++i)
        scrinx[i]->fld_attr = scrinx[i]->type = 0;
    scrinx[23]->hdg_attr = X_CLEAR;
reset:
    page = first;
    prev = NULL;
    goto samepage;
nextpage:
    page = this;
    for (j = 18; j; --j)
        if ((prev = links[j]) != NULL)
            goto samepage;
    page = first;
samepage:
    that = this = page;
    for (j = 1; (j < 19); ++j)
    {
        links[j] = this;
        strnpad(scrinx[j]->data,
            ((this != NULL) ? this->line : "\0"),79);
        if (this != NULL)
            this = this->link;
    }
    firsttime = 'f';
    if ((key = inkeyboard(stdscr,params)) != KEY_F(9))
    {
        for (i = 1, j = 0; i < 19; ++i)
        {
            scrinx[i]->type &= ~X_CURSOR;
            if (that != NULL)
            {
                strncpy(that->line,scrinx[j - 1]->data,79);
                that = that->link;
            }
        }
        for (k = ++j; k < 19; ++k)
            if (blanks(scrinx[k]->data))
            {
                for ( ; j < 19; ++j)
                {
                    if ((that = (struct groups *)
                        calloc(1,sizeof(struct groups))) == NULL)
                        goto termin;
                    ++recs;
                    strncpy(that->line,scrinx[j]->data,79);
                    links[j] = last->link = that;
                    last = that;
                }
                break;
            }
    }
    switch (key)
    {
    case KEY_F(2): /* PF2 insert line */
        if ((links[j = params->last_field] != NULL) && j)
        {
            if ((that = (struct groups *)
                calloc(1,sizeof (struct groups))) == NULL)
                goto termin;
            ++recs;
            if (links[j] != first)
                ((j == 1) ? prev : links[j - 1])->link = that;
            else
                first = that;
            that->link = links[j];
            if (j == 1)
                page = that;
            scrinx[j]->type |= X_CURSOR;
        }
        goto samepage;
    case KEY_F(4): /* PF4 save file */
        if ((text = fopen(filename,"w")) != NULL)
        {
            for (this = first, valid = NULL; this != NULL;
                this = this->link)
            {
                this->line[79] = 0x00;
                if (blanks(this->line))
                    valid = this->link;
            }
            for (this = first; this != valid; this = this->link)
            {
                *(this->line + blanks(this->line)) = 0x00;
                if (*this->line)
                {
                    fputs(this->line,text);
                    putc('\n',text);
                }
            }
            fclose(text);
            goto termin;
        }
        beep();
        goto samepage;
    case KEY_F(5): /* PF5 delete line */
        if ((links[j = params->last_field] != NULL) && j)
        {
            --recs;
            scrinx[j]->type |= X_CURSOR;
            if (first == last)
                first->line[0] = 0x00;
            else
            {
                that = links[j]->link;
                if (links[j] != first)
                    ((j == 1) ? prev : links[j - 1])->link = that;
                else
                    first = that;
                if (links[j] == last)
                    last = ((j == 1) ? prev : links[j - 1]);
                if (j == 1)
                    page = links[2];
                links[j] = NULL;
            }
        }
        goto samepage;
    case KEY_F(7): /* PF7 top of file */
        goto reset;
    case KEY_F(9): /* PF9 undo changes */
        goto termin;
    default:
        goto nextpage;
    }

termin:
    termscreen();
    exit(0);
}

blanks(s)
char *s;
{
    short int i = 0, j = 0;

    while (*s)
    {
        ++j;
        if (*s != 0x20)
            i = j;
        ++s;
    }
    return i;
}

strnpad(d, s, n)
char *d, *s;
short int n;
{
    while (n--)
        *d++ = (*s ? *s++ : ' ');
    *d = 0x00;
}
```

ADA® And The 68000

Theodore F. Elbert
The University of West Florida
Pensacola, Florida 32514

Part-10 Ada Compilers and Software

In the concluding part of this series, a review of the requirements of an Ada compiler will be presented, together with a brief summary of the status of various Ada compilers, development projects, and other software.

In Part I, the standardization of the Ada Language was discussed. It was noted that two approaches to standardization were used:

- Specification of the Ada Language both as s military standard (MIL-STD) and as a standard of the American National Standards Institute (ANSI).
- Registering of the term "Ada" as a trademark of the United States Department of Defense.

... guidelines established by the Department of Defense, the mark "ADA" shall not be used to name or identify o compiler or language translator unless it complies fully...

According to guidelines established by the Department of Defense, the mark "Ada" shall not be used to name or identify a compiler or language translator unless the compiler or translator fully complies with the Ada standard (ANSI/MIL-STD-1815A-1983). Furthermore, such compliance can only be demonstrated by having the compiler or translator satisfactorily pass more than two thousand tests contained in a so-called "validation suite" of programs. Validation is administered by the Ada Joint Program Office at one of three validation test sites established expressly for this purpose.

To ensure that all validated Ada compilers meet contemporary minimum standards, the Ada Joint Program Office validates compilers only for a one year period. That is, Ada compilers must be validated annually. To further complicate the situation, the test suite of validation programs is modified on a continuing basis as the language matures, and as more experience is gained in validation. Thus, a compiler that was validated last year may become an unvalidated compiler this year if it cannot pass the current validation teats. In fact, some previously validated Ada compilers have suffered this fate, probably because economic factors could not justify the additional investment required to bring them into conformity. Currently (late 1985), there are fourteen validated Ada compilers. These compilers are listed below, with the host and target machines indicated.

COMPANY	HOST COMPUTER	TARGET COMPUTER
Alsya	VAC(VMS) MicroVAX	Altos 68000
Data General	D.G. Computers	D.C. Computers
DEC	VAX(VMS) MicroVAX	VAX(VMS) MicroVAX
Honeywell	Honeywell Computers	Honeywell Computers
Rational	Rational Development Environment	Rational Development Environment
ROLM	ROLM Computers	ROLM Computers
Telesoft	VAX(UNIX)	VAX(UNIX)
Telesoft	VAX(VMS)	VAX(VMS)
University of Karleruke (West Germany)	SIEMANS	SIEMANS
University of Karlsruke	VAX(VMS)	VAX(VMS)
SOFTECH	VAX(VMS)	VAX(VMS)
Verdix	VAX(UNIX)	VAX(UNIX)
Verdix	Sun	Sun
Verdix	VAX(ULTRIX)	VAX(ULTRIX)

More than just these fourteen compilers have been validated, but some have expired without revalidation.

There are several interesting features of this list:

- Two of the fourteen compilers are European.
- The VAX seems to be a popular host computer.
- There are very few realistic target computers listed. (The Data General, Honeywell, and ROLM computers are exceptions.)
- There are virtually no microcomputer based compilers. (The SUN workstation is the exception; it uses the 68000 family of devices.)

Moat of these compilers have been developed by private industry; some have been developed under government contract. In addition, it should be mentioned that these systems are not cheap. Purchase of one of these compilera will run from $20,000 up. The Rational Development Syatem (including hardware) costs just under $500,000.

In addition to the compilers that are currently validated, there are many presently under development, including several that are specifically targeted to microprocessor based computers. These development efforts are listed below:

COMPANY	HOST COMPUTER	TARGET COMPUTER
Alsys	68000,IBM-PC/XT	68000,IBM-PC/XT
Amdahl	Amdahl 470	Amdahl 470 8086
Bell Laboratories	VAX 3B20	VAX 3B20
Carnegie-Mellon University	VAX	VAX
Control Data	CYBER	CYBER
Digicomp	Delphi	Delphi
Florida State University	CYBER	Z8000
General Transformation	IBM-PC/XT	IBM-PC/XT
Gensoft	VAX WD1600	VAX WD1600
Intel	8086 80186 80286 80386	8086 80186 80286 80386
Intermetrics	IBM-370 Prime	IBM-370 Prime
Irvine Computer Science Corp.	VAX 68000 Z8000	VAX 68000 Z8000
Mills International	Burroughs	8086 Z80
New York University	VAX IBM-370 IBM-PC/XT Sun	VAX IBM-370 IBM-PC/XT Sun
R R Software (Janus-Ada)	8086 MS-DOS 8086 CP/M	8086 8080 Z80
Softech	VAX	8086 80186 80286 MicroVAX
Stanford University	DEC-10	DEC-10
Tartan Laboratories	VAX(UNIX) 68000(UNIX)	68000 68010 68020
Telesoft	IBM-370 68000(UNIX) 68000(ROS)	IBM-370 68000
Verdix	VAX Sun	68000 1750A 80286

In addition, there are a number of efforts currently underway in Europe and Japan. The general trend indicated in these tables is toward development of Ada compilers that are targeted to the host computers themselves, which seems reasonable at this point in the development of Ada language capabilities. The next logical step is to re-target these compilers to more reasonable target machines. Some movement in this direction can be seen in the various efforts to target the Intel (8086, 80186, 80286, 80386) and Motorola (68000, 68010, 68020) families of advanced microprocessors.

As part of its promotion of the Ada language, the Ada Joint Program Office has established the Ada Information Clearing House (Ada IC) to aid as a focal point for information on the development of the Ada program and on all Ada language related activities. Ada IC is operated by IIT Research Institute and provides information on program status, AJPO activities, Ada standards, educational programs, and European Ada activities. Specific activities of the Ada IC include:

- Publication of the Catalog of Resources for Education in Ada and Software Engineering (CREASE).
- Maintenance of a current data base of Ada-related information. This data base is accessible via MILNET (formerly ARPANET) and via TELENET.
- Publications of the Ada IC Newsletter.
- Maintenance of an Ada mailing list database.
- Provision of an Ada Information Packet (free of charge) to interested parties.

In general, a wealth of introductory information is available from Ada IC. Two current addresses for Ada IC are:

3D139 (Fern St., C-107)
The Pentagon
Washington, D.C. 20301-3081
(703) 685-1477

and

Suite 300
4550 Forbee Blvd.
Lanham, MD 20706
(301) 731-8894

In the experience of some, a phone call is often more effective than a letter.

The professional organization most oriented to Ada language is the Special Interest Group on Ada (SIGADA) of the Association for Computing Machinery (ACM). This group publishes Ada Letters, a bi-monthly journal containing both technical articles and important information relating to Ada language development. The quarterly meetings of this organization provide the principal forum for the exchange of ideas among Ada experts.

This series of articles has presented the Ada programming language from three viewpoints:

- The development of the language and the impetus behind that development.
- The general form of the language in terms of programming features.
- The current status of the Ada language.

While no one knows what the full impact of the Ada language will be, it cannot be denied that the current situation is dynamic and exciting. Defense contractors

are currently "tooling up" for the full implementation of Ada in their facilities, and the close relationship between the Ada language and modern concepts of software engineering is beginning to emerge in the form of integrated workstations and support environments. Critics may argue against the wisdom of the language, but they cannot deny its impact.

®Ada is a trademark of the U.S. Department of Defense.

. . .

Editor's Notes: I want to thank Dr. Theodore Elbert for the time and effort he has given to provide us with this series on Ada. When he first submitted the original Ada manuscript I asked him to do more, he did - result: a comprehensive and complete "over-view" of the Ada language.

Without the 'spirit of cooperation' as evidenced by this and other contributions to 68 Micro Journal, we would be hard put to justify our existence. So, what it all simply boils down to is this: "Without all of you, we would be nothing!" You, the readers, are 68 Micro Journal.

So to you Theo, and to all the other who contribute, for the enlightenment of us all, I and thousands of readers say - **THANK YOU, PLEASE KEEP THE GOOD STUFF COMING!!**

DMW

68000 User Notes

By: Philip Lucido
2320 Saratoga Drive
Sharpsville, PA 16150

All right, it seems that I missed another month. But hear me out - it wasn't burnout this time! As a matter of fact, I've been getting a fair amount of use out of my computers. I've even bought a new piece of software - a terrific implementation of APL for the Macintosh from a company called Portable Software.

So why the missed column? Well, it seems the company I work for contracted to do a large project for another company about a year ago. Seems I badly underestimated the time required to do the project. Seems I just finished spending about five weeks, working seven days a week, up to 15 hours a day, just to get the project done by the mid-November deadline. Seems I had no time for other activities, like writing columns, eating, or sleeping.

An aside to people engaged in writing software professionally - when asked how long I think it will take to finish some project, I make my best guess, double that, add 10 or so (be it hours, days, or months), then triple the result. It would appear that my formula gives an answer on the low side. Does anybody out there know a better way?

My Very First Program

Enough with the excuses. Onward! The last column was an introduction to the binary and hexadecimal numbering systems. While not exactly as thrilling as, say, watching grass grow, it was necessary. Now, though, I can start with some actual programming and hands-on usage of the OS-9 assembler and debugger.

Writing a program in assembly language involves a number of steps. Humans cannot easily make sense of strings of numbers, while computers can, in some ways, make sense of nothing else. To allow the two to communicate, some intermediate processing is required. An assembly language program, in human-readable form, is a text file known as the source program. This source text is input to the assembler, a program which converts the text into a form more in line with the computer's requirements. The output from the assembler is called the object file. In some cases, this object can be directly executed, so no more steps are required. This is the case for OS-9 on the 6809. For the 68000, though, a further step is required. Here, the assembler produces a relocatable object file, which must be run through a linker, producing a final executable object file. The linker is used to allow separate source files to refer to the contents of each other, and to allow your programs to use pre-written segments of code available in a type of object file called a library. We'll have reason to use these capabilities of the linker in later columns.

Humans cannot make sense of strings of numbers, while computers, in some ways, can make sense of nothing else.

Once we have an executable program, we can use it just like any other program under OS-9, such as dir or copy. Usually, though, a program will not work properly on the first try. In this case, a program known as the debugger is used. The debugger allows us to examine a test program while it runs, looking for errors along the way. While the 6809 OS-9 debugger was rather bare-bones, the 68000 debugger is very powerful, allowing a test program to be run a step at a time. ■■

Enough intro - on with an example. Along with this column is a teat program. Type it in, using the name prog001.a. The suffix .a means that this is an assembler source file.

What's It Look Like?

Before we try the program out, let's look at it a little more closely. Each line in an assembly source file is a separate statement. There are three different kinds of statements. First, there are comments, which are there for the people reading the source, and are ignored by the computer itself. In this assembler, the comment lines are those beginning with an asterisk in column 1, as well as the lines which are blank. Next, there are the instructions, which describe the actual steps to be taken in the final executable program. Each such line assembles to a single step, as opposed to other languages like C or BASIC where each statement can translate into a large number of steps when a program is run. In the example, the instructions are those lines from the one starting with "start:" to the one starting with "done:". Finally, there are directives, which hold directions for the assembler. These lines look much like instructions, but change the way in which the assembler looks at your source instead of describing a step in the final program. In the example, the lines with "psect" and "ends" on them are directives.

Instructions and directives must follow a certain syntax to be understood by the assembler. Each such statement consists of from one to four fields, or line sections separated by spaces or tabs. The four fields are label, operation, operand, and comment.

The label field must start in the first column. It is usually optional, but may be required on some lines, forbidden on others. If a statement does not have a label field, it must start with a space or tab character. The label itself is an example of a symbolic name, or aymbol. A symbol is just a name starting with a letter, followed by zero or more letters or digits. For the OS-9 assembler, a symbol may be up to nine characters long, and may also use the special characters "$" or "_", except as the first character. Also, upper and lower case are not treated the same, so the symbols "ABCDE" and "Abcde" are distinct.

When a program is assembled, each instruction statement will produce code which will reside at a certain address in memory. The label field is a way of associating a name with that address, so other statements can refer back to that address without knowing what it's final value will be. A label is something like a line number in BASIC, and is exactly the same as a label in C. In the example, the labels are "start", "loop", and "done". Note that all three labels have a colon following them. The colon is not part of the label symbol, but instead tells the OS-9 assembler that the actual label names are to be included in the object file, so the linker or debugger can refer to addresses by name, just like lines within the source.

Next comes the operation field. This field determines what the line will actually do. In an instruction, the operation field holds an opcode, or op for short, which describes the type of step to be executed. The assembler converts the opcode, along with the operand field, into code values which are directly executed by the 68000 as a single step in the final program. In a directive, the field holds a psuedo-op, which, aa the name suggests, does not describe a true step in the executable program, but describes an operation to be understood by the assembler instead. The opcodes and psuedo-ops are sometimes known collectively as the mnemonics, since they are generally short abbreviations which are mnemonic names for the operations they represent. Finally, note that in the example, some of the opcodes end in ".s". This suffix is a size attribute required in 68000 programs. Most 68000 instructions can operate on different sizes of data, either 8, 16, or 32 bits at a time. By default, 16 bit data is used, but a size suffix appended to the mnemonic causes another size to be used instead. I'll hold off describing these further till later.

The operand field is next, following the operation field. While an opcode describes the operation to be performed, the operand describes the values to be used in performing the operation. For instance, if the opcode is "add", then the operand will describe the two numbers to be added together, as well as the location to save the result. The operand field can be very complicated, since the 68000 has many ways of determining where to find the data for an operation. For some opcodes, though, no extra data is needed to determine how the operation is to be done, so no operand field is necessary.

The final field in a statement is the comment field. As with comment lines, this is simply text meant for any human who happens to read the source. It is ignored by the assembler. In the OS-9 assembler, the comment field is anything following a space or tab which ends the operand field, or the operation field if the mnemonic does not require an operand. Be careful - in many other assemblers, the comment field must begin with a special character, usually a semicolon.

What I've described as the assembly language source file pretty much applies to all assembly languages, not just the 68000.

What's It Do?

What I've described as the format of an aasembly language source file pretty much applies to all assembly languages, not just the 68000. The differences from processor to processor are in the mnemonics used in the operation field, and the various forms allowed in the operand field. Let's examine the teat program to get some idea of what goea on in the case of the 68000.

The first non-comment line in the example has an operation field of "paect". This is a psuedo-op for an assembler directive, which is required for OS-9, not the 68000. I'll ignore it for now; just enter it as is. The first instruction statement is the one with a label of "start". The opcode is "move" and the operand is "#0,d0". The move opcode, as you'd expect, moves data within the 68000. When you move something, you take it from a starting point, the source, and copy it to an ending point, the destination. The source and destination to be used are given in the operand, source first, then a comma, then the destination. For this instruction, the source is "#0", and the destination is "d0". This operand

form of "source,destination" applies to many 68000 instructions, not just move.

The source for the current instruction is "#0" and the destination is "d0", but what does this really mean? Well, data in a 68000 can exist in one of two places - either in memory or in a register. Memory is just a long array of locations, which the 68000 can read or write by giving the proper address. Each memory location is a single byte (a collection of 8 binary bits) but bytes located next to each other can sometimes be treated as single 16 bit or 32 bit quantities. A register is a sort of super-powerful memory location, which the 68000 can address much faster than normal memory. This is because a register is actually located within the 68000 microprocessor chip itself, while normal memory is located in separate chips. Most manipulation of data in a program is done in registers, since registers are so much faster than normal memory, and the 68000's instruction set allows many more things to be done with registers than with memory.

In the 68000, each byte of memory has a numeric address. These addresses range from $000000 (zero) to $FFFFFF (about 16 million). The numbers are rarely used, though. Instead, a label is normally used. The registers in the 68000 do not have numbers to address them. Instead, they have names. There are 16 main registers, with the names D0, D1, D2, D3, D4, D5, D6, D7, A0, A1, A2, A3, A4, A5, A6, and A7. D0 to D7 are the 8 data registers, used for most general arithmetic. A0 to A7 are the 8 address registers, used to hold addresses which refer to places in the address array.

We now have enough to understand the move instruction. The source of "#0" means that the number zero is the value to move. The pound sign indicates that the zero is the actual value, not the address in memory where we want to get the value. The destination of "d0" means that data register 0 is used to store the value. In other words, this instruction puts a zero in data register 0.

The rest of the instructions follow easily. "move #1,d1" has the same form, and puts a one into data register 1. For "add d1,d0", remember the "source,destination" form. This instruction takes a value from data register 1 and adds it to the value in data register 0, storing the result back in data register 0. The "add" opcode, like many others, actually takes two values, does some operation on them to arrive at a single result, and puts the result back somewhere. While this involves three different places (two sources, one destination), the 68000 requires one of the sources to be the same as the destination.

The next instruction, "add #1,d1" has the pound sign form for the source. Here, the value one is added to the value in D1, the result being placed back in D1. Next comes "cmp #100,d1". Cmp is short for compare. In a compare, we have two sources, but no real destination. Instead, some flags are set somewhere else (the condition codes - never mind what that means for now), to be used by a later instruction. The compare instruction here checks if the value in D1 is less than, equal to, or greater than the value 100.

"ble.s loop" uses the result of the comparison. Ble stands for "branch on less than or equal to". This is an example of a conditional branch. Depending on certain conditions, determined by the type of conditional branch, the next instruction to be executed is either the next instruction in memory (the branch is not taken), or the next instruction is located at an address given by the operand field of the branch (the branch is taken). For this instruction, the operand is "loop". If the comparison shows that the value in D1 is less than or equal to 100, then the next instruction to be executed is the line with the label field of "loop".

If the branch is not taken, then execution proceeds with the next line, the one with the label "done". The opcode, "bra.s", is an unconditional branch, which causes execution to proceed at the line indicated by the operand field, no matter what the condition. Since in this case the operand field says "done", the same as the label field, this is an example of an infinite loop. This instruction will be repeatedly executed, until you decide to stop the program by using the abort key under OS-9. With a less powerful operating system, this program, as is, would cause the entire system to hang, with only a full reset freeing the machine.

Now that we've seen each individual instruction, go back over the program as a whole. The comments say that the purpose of this code is to add the numbers from 1 to 100 together. This requires two different values. One value holds the current number, which moves from 1 up to 100. The other value holds the working total, which will be the final total when the first value passes 100. In the example, the counter value is held in D1, and the total in D0. Each time we execute the four instructions starting at label "loop", we add the next value in the D1 counter into the working total in D0. Thus, when the loop is finished because the value in D1 reaches 101, the final sum will be in D0.

Having gotten this far, I'm out of space to actually show this program in action. Since the program doesn't actually output anything, and ends with an infinite loop, it is only suitable for running with the debugger. That, unfortunately, will have to wait until next month. Hang in there, things will pick up presently. In the meantime, you have a whole month to enter the example program. Presumably, that will be sufficient. See you next month.

** Editor's Note: Please note that S.E. MEDIA is now offering (at a large intro discount) a single stepping, etc., first class debugger - SOLVE, for OS-9 6809, levels I & II only. It is very powerful but simple to use! This has proven to be a real popular item. IF you don't consider it now, at it's low intro price, you may certainly regret it later!

DMW

\- - -

```
* 68000 User's Notes - Program #001

        psect   Prog001,$0101,$8001,1,512,start

* This is a small test program.
* It adds the numbers from 1 to 100.

start:  move    #0,d0       start total
        move    #1,d1       start next number to add
loop:   add     d1,d0       add in the next number
        add     #1,d1       increment value to add
        cmp     #100,d1     check if up to 100 yet
        ble.s   loop        no - continue looping
done:   bra.s   done        done - sit here in loop

        ends

* End of program #001
```

Basically OS-9

Ron Voigts

I SPEAK OS-9

A few months ago I received a question from an OS-9 user who was getting started. He had created a number of files that had similar names like LETTER1, LETTER2 and so on. He wanted to see the contents of the files, but didn't want to have to enter the word processor to do it. So he entered the file's name in response to the OS-9 prompt and got an error #216 which is Path Name Not Found. He knew he had saved and loaded files before with the word processor, so they must be there. His problem was he didn't know how to talk to OS-9.

Through out the years of grade school and high school we have been drilled on proper english. We are taught, there are particular rules of grammar that have to be followed. Complete sentences must be made consisting of a subject and a verb. As well as, other parts of the sentence must be used to make its meaning more complete. OS-9 is no different.

When you want to talk to OS-9, you use sentences too. Conveniently, it provides the subject in the form of a prompt that looks like:

```
OS9:
```

All you have to do is supply the verb or more formally called the command. When you want to look at the working data directory, enter:

```
OS9:dir
```

Now you have a complete OS-9 sentence that will print a list of the files in the current working data directory.

Many commands take optional parameters, some demand one or more parameters to make the OS-9 sentence command complete. In the case of our DIR command, we could have tried:

```
OS9:dir e
```

The little "e" tells DIR to make an entire description of the files. A file's name, size, address, owner, permissions, date and time of last modification are listed. While some parameters are options, others are necessary for the command. If you want to see the contents of a file, entering:

```
OS9:list letter1
```

will cause the contents of "letter1" to go the screen. This is what our OS-9 user should have used to examine his files. The OS-9 COPY command needs at least two parameters. It needs a a source file and a destination file. To use it you would enter:

```
OS9:copy /d0/afile /d1/afile
```

and you would be copying "afile" from /d0 to /d1. COPY gives you another option which is "-s". Entering:

```
OS9:copy /d0/afile /d0/file -s
```

would let you copy "afile" to another disk while using the drive /d0. At each pass, a prompt will be displayed telling what disk should be in drive /d0. Parameters will vary from module to module. Some will be mandatory, while others will be optional. It pays to know the command well and what it can do.

Another part of talking to OS-9 is the use of modifiers. Modifiers are a part of the shell. They change the way the shell would otherwise execute the command. They either come just before the parameters or right after them. There are 6 and they are:

#	memory size
!	use with pipes
<	redirect standard input
>	redirect standard output
>>	redirect standard error output
&	run as a background task

We've hit on many of these in past columns, like the standard paths.

To redirect the input and output from the terminal, you use ">, < or >>". So to send a listing the printer, enter:

```
OS9:list file >/p1
```

and the listing will go the device p1. Many times when debugging a program that is to get its input from the keyboard, I create a file with the necessary information and redirect the standard input. For a recent program at school that analyzed flow through a complex piping network, I used:

```
OS9:paacaln flow <pipes
```

Here "flow" was a pcode program and "pipes", the temporary file. Also the error path can be redirected. Again back to debugging "flow". It was written in Pascal, so when an error did occur, at least half a dozen lines of error code would appear on my crt screen. There was the error number and type, the calling procedures, the line number and the pcode number. Rather then letting it print to the terminal I would enter:

```
OS9:pascaln flow <pipes >>/p1
```

I could then take the computer error listing, a copy of the source code and more carefully look for the error.

The "#" is a way to tell OS9 how much memory you want to use. At run time a module is allocated memory by what it needs. Some processes can use more memory, if it ia available. Usually these are ones that use a buffer area for data manipulation. Editors and word processors are good examples. Assemblers and compilers are another. Many of the OS-9 commands that involve I/O can utilize the extra memory. A good example is the COPY command. It normally requests 4K of memory. If you are copying a file that is say 20K long, it would take at least 5 passes or more. But if you enter:

```
OS9:copy /d0/file /d1/file #22k
```

you provide for 22K of buffer area and the copying may only take a pass or two. You could have also used:

```
OS9:copy /d0/file /d1/file #88
```

This tells the shell to use 88 pages of memory. At 256 bytes per page, that comes to 22k again.

The "!" is used with pipes and leta you pipe the output of one process to the input of another. Let's aay you have a module called UPPER. It excepts from the standard input characters, converts them to uppercase if they are not already and prints them to the standard output. Another module is called STRIP. It inputs also from the standard input, strips out linefeeda and prints to the standard output. You have a file with linefeeds

in it and is mixed with upper and lower case. You type:

```
OS9:list file ! upper ! strip >/d1/newfile
```

"newfile" will be on drive /d1. Everything in "newfile" will be in uppercase and the linefeeds ($0A) will be removed.

The final modifier is "&". Remember part of the power of OS-9 is to handle multitasking. Many times I will list files to the prioter as a background task, while I work on something else. Entering:

```
OS9:list somefile >/p1& 
6004
```

will send a listing of "somefile" to the printer. Meanwhile, I'll edit another file on the disk. The 6004 tells you that this is process #4. If for some reason you want to stop it, you'd enter:

```
OS9:kill 4
```

A built in shell command, KILL, removes it from the process queue.

This gives a fairly good overview of how to talk to OS-9. Being able to effectively use the OS-9 commands, opens up the power of OS-9. Many other systems I have worked on, I've lamented, if only I could do this or that. With OS-9 the power was there. All I had to do was use it properly.

BasicO9 From OS-9

Variations brought about by the different programming languages can influence how the OS-9 command line is written. BasicO9 is a good example. Once a program in BasicO9 has been packed and placed in the commands directory, it can be used as a regular command. But its syntax is s little different.

The command is entered much as the module would be run from a BasicO9 program except that RUN is not placed in front of the module name. Within a BasicO9 program, a module called DOIT might be used with the line:

```
run doit
```

but from the shell level, it would only be necessary to type:

```
OS9:doit
```

OS-9 would deduce that this is a BasicO9 module, get RUNB from the commands directory and execute DOIT. If parameters are to be passed, they are included as arguments with the the module name. So to pass some type of information to DOIT, the entry might appear:

```
OS9:doit(5,"test")
```

The first parameter is the integer 5. The second is the string "test". The number of parameters, their order, size, shape and type must agree with what is expected by the module. In our example it is DOIT.

DOIT must be written to accept the parameters. This is done with the statement, PARAM. To pass the integer and string we used before, DOIT might have a few lines at its beginning that look like:

```
PARAM number:INTEGER
PARAM word:STRING[10]
```

Notice that string is dimensioned for 10 characters and "test" was only 4 long. This doesn't violate the rules since BasicO9 strings can be shorter than the space provided.

This month's program is a BasicO9 one that gives you a chance to try out passing parameters. It is called FLOOK which is short for File LOOK. Its syntax is:

```
OS9:flook("parameters","path")
```

The parameters and their use are:

```
n....number lines
w....word count
c....character count
l....line count
/....search for target that follows
```

The path can be any to a file. Let us say you want to look for a string, "hello" in a file called letter on drive 1 in the directory, MARCH. Further, you want to number lines that the target word is found on. So you would enter:

```
OS9:flook("n/hello","/d1/march/letter")
```

The output would be from the file, letter. The lines printed would have "hello" in them and be numbered as to their position in the file. You can also use the program to "pretty print" a C program. Just enter:

```
OS9:flook("n","prog.c")
```

and prog.c will be printed with line numbers added. Maybe you want the vital statistics on a file, then try:

```
OS9:flook("wlc/xyz","another_file")
```

You'll find out how many word, letters and characters your file has. As long as the target, "xyz", is not found there will be no listing.

There are a few enhancements you can add. For example, a parameter could be added that would suppress the printing of the file all together. Then a silly target string would not have to be passed. Also a "head" or "tail" function could be added that would let FLOOK print only the the first or last few lines of the file. I'll leave these enhancement up to you.

The best way to learn a foreign language is to speak it. The best way to learn OS-9 is to sit down at the keyboard and start typing commands. You might end up with a lot of error messages, but remember, it knows how to listen, you have to learn how to speak to it.

```
PROCEDURE flook
(* BasicO9 Program:  FLOOK
(* By: Ron Voigts    Date: 9-Nov-85
(* Usage: flook( "modifier string", "readfile")
(*
(* This program will list a file, adding line numbers.
(* It will also search for a target string.
(* And it will count characters, words and
(* lines used.
(*
PARAM modifiers:STRING[80]
PARAM pathname:STRING[80]
DIM line:STRING[132]
DIM linecount,wordcount:BOOLEAN
DIM charcount,numberit:BOOLEAN
DIM search,inword:BOOLEAN
DIM c:STRING[1]
DIM target:STRING[80]
DIM number,words:INTEGER
DIM characters:INTEGER
DIM i:INTEGER
DIM path:BYTE
*
(* Set the flags and parse the parameter list
linecount:=FALSE
wordcount:=FALSE
charcount:=FALSE
numberit:=FALSE
search:=FALSE
WHILE LEN(modifiers)>0 AND LEFT$(modifiers,1)<>"/" DO
  c:=LEFT$(modifiers,1)
  IF c="N" OR c="n" THEN
    numberit:=TRUE
  ENDIF
  IF c="W" OR c="w" THEN
    wordcount=TRUE
  ENDIF
  IF c="L" OR c="l" THEN
    linecount:=TRUE
  ENDIF
  IF c="C" OR c="c" THEN
    charcount:=TRUE
  ENDIF
  modifiers:=RIGHT$(modifiers,LEN(modifiers)-1)
ENDWHILE
IF LEFT$(modifiers,1)="/" THEN
  search:=TRUE
  target:=RIGHT$(modifiers,LEN(modifiers)-1)
```

```
ELSE
  target$=""
ENDIF
!
!* Open the file, read in lines from the text
!* and process them.
number=0
characters=0
words=0
OPEN #path,pathname$:READ
WHILE NOT(EOF(#path)) DO
  READ #path,line$
  number=number+1 \REM Count lines
  characters=characters+LEN(line$)
  inword=FALSE \REM count words
  FOR i=1 TO LEN(line$)
    IF MID$(line$,i,1)=" " THEN
      inword=FALSE
    ELSE
      IF inword=FALSE THEN
        inword=TRUE
        words=words+1
      ENDIF
    ENDIF
  NEXT i
  IF search$ THEN \REM Print target lines
    IF SUBSTR(target$,line$)<>0 THEN
      IF numberit THEN
        PRINT USING "#5^,s",number," ";
      ENDIF
      PRINT line$
    ENDIF
  ELSE \REM or print all lines
    IF numberit THEN
      PRINT USING "#5^,s",number," ";
    ENDIF
    PRINT line$
  ENDIF
ENDWHILE
CLOSE #path
!
!* Finish with necessary line, word and
!* character counts
PRINT
IF linecount THEN
  PRINT USING "Lines read: #,#5^",number
ENDIF
IF wordcount THEN
  PRINT USING "Words read: #,#5^",words
ENDIF
IF charcount THEN
  PRINT USING "Characters read: #,#5^",characters
ENDIF
END
```

MUSTANG - 020

"C"
"Real" Life Test

For the past few months you might have noticed that I have not said a lot concerning our MUSTANG-020 68020 system. Despite it being the fastest system for anywhere near the money of ANYTHING presently available.

Also you will be seeing a lot of "benchmarks" of this system as opposed to others. Every day we run all sorts of neat testing routines. However, of all those we have used or published, I have yet to include one in any programming I have done. They are just not 'real life' type procedures. They are fine for comparison, but what I (and you) really want to know is - how does it evaluate to "daily" usage? In other words, "what does it really do, doing the things you and I do daily with a computer? So, just for you and me, I conducted the following, very unscientific "benchmark" series.

The K & R Benchmark
(sorta)

I extracted from K&R the "list" program. The changes I made were to replace the "printf" function with "puts", see listing below. Also for the 68020 I declared 'c' to be "register", but not "long", I wanted to give the others fair "shake" on this test. Otherwise it is fairly 'stock K&R'.

The times indicated below are pretty well what I expected. The more I/O you do, the slower the operation. Note that the floppy times for both the level II 6809 and floppy 68020 are very close. Surprised? Well you shouldn't be, practically all the time is spent reading and writing to the disk. So, it make little difference how rapidly the CPU handled the code, it still sat around a relatively long time waiting for disk reads and writes. This is very pronounced for short source programs and compilers. The compiler must load and unload the same data for floppy or ramdisk operations. However, as the code gets longer (or more complex) the differences in speed becomes more apparent.

For a source program of 240 lines as opposed to our very simple list program, the difference gets real one-

As more time is spent computing code and less time doing I/O, the real advantage of the 68020 becomes very apparent !!

sided. As more time is spent computing code and less time doing I/O, the real advantage of the 68020 (or any more efficient CPU) becomes very apparent!

... if you do anything beyond the very simple, the Mustang -020 ... is far Superior !!!

** Three New Options **

For example the 240 line program (no fancy math) mentioned above still compiles, on the MUSTANG-020 68020 system, in RamDisk, in about 4.3 seconds. Less than twice the time but about 12 times more code and work. THe point being, "the larger and more complex the task, the more efficient the better CPU". The spread continues to expand as the task becomes larger or more complex.

So here are my results, done in a very unscientific manner. But they do reflect what I and most of you do, reasonable things, not too simple and not overly complex (on the surface, that, is).

6809	1 mhz	OS9 level I	floppy	2 milsec step	3 min 28 sec
"	2 "	" "	II "	" "	1 min 49 sec
"	"	" "	HardDisk		44.2 seconds
"	"	" "	RamDisk		28.4 seconds
68020	12.5 mhz	OS9 68K	floppy	2 milsec step	1 min 03 sec
"	"	" "	HardDisk		18.6 seconds
"	"	" "	RamDisk		2.8 seconds

The systems used:

6809 Level I OS9	Sardis SBC (CoCo OS9)
6809 Level II OS9	CIMIX III
68020 Level 1 OS9 68K	MUSTANG-020

All three systems were running the latest (we have) of the Microware 'C' compiler. The source was unchanged from system to system.

The Level I's much slower times are not completely a fault of the compiler or the CPU. Because of memory constraints, the compiler, in this version is in two parts. Actually this (Sardis) is a more efficient Level I than most other Level I systems (more RAM available). Also more temporary files are required to be built (and deleted) by all Level I systems. Actual CPU times were much closer. But nevertheless, it still took far more 'real time'. And that is what **REAL**ly counts - **REAL TIME!!**

So if you do anything beyond the very simple, the MUSTANG-020 68020, with a full 2,000,000 plus bytes of RAM is far superior.

Compared to several 68008 and 68000 systems we have tested, the **MUSTANG-020** is still as much as 18-20 times faster. Which means that your 6809 will even 'whip-up' on them (68008/68000) in many applications. But for real 68XXX power, the 68020 is, by far, **KING OF THE HILL!**

Some More New Stuff

By the time this gets to you, we should be fairly well into our port expansion options. As of now there will be three options, not including the standard pre-wired 4 serial port card and cable, which is included with all Mustang-020 systems.

The first is an additional 8 port expansion pre-wired, standard DB25 connectors. This consists of the main board adaptor, cable and two additional pre-wired four port serial cards. You can inatall more than one of these. As all options - no soldering or wiring - everything pre-wired, burned-in and ready to plug on.

Also coming in the next 60 or so days is an expansion card containing DB9 serial connectors, pre-wired with adaptor connector, dc-dc converter and all necessary hardware. The pin-outs are identical to the IBM AT. This means that if you want to hook up a "local computer store" serial anything (that works on the IBM AT) you can buy IBM AT cables, and say a modem, for instance, simply plug them into the MUSTANG-020 DB9 connectors, and go.

We have tried to take all the hassle out of bringing up or using this system. We (CPI, Data-Comp) listen to what you tell us! That is why we have packaged up for you our readers, the most powerful system available, and at prices others can't touch! **The MUSTANG-020, from Data-Comp, is the most powerful, and least expensive system available -ANYWHERE!!!!!!!!!!!!!!**

Because of the various options, we will be delivering several different cabinet configurations. Each designed to accept additional ports, boards (expansion RAM, CAD, etc.) allowing future expansion of the system.

The basic system, consisting of the main board, hard disk controller board, winchester (1), floppy (1) and all cables and power supply (switching), and standard cabinet (M020 cabinet). IF you think that you will be expanding the system after delivery, then I suggest you consider starting out with one of the larger cabinets. The price difference between smallest to largest presently available cabinets will be less than $100. Just though you might like to know.

And of course, as is the Data-Comp promise to try to make it the best they can for our readers - you can buy any or all parts as you need them.

DMW

- - -

```
/* A C listing program - very simple and basic */
#include <stdio.h>
main (argc,argv)
 int argc;
 char *argv[];
{

register char c;
FILE *fl, *fopen();

if (argc !=2){
  puts("\n Useage:  $ list filename");
      exit (1);

       }

if ((fl = fopen(argv[1], "r")) == NULL){
            puts("Can't open called file");
        exit (1);
}

while ((c= getc(fl)) != EOF)
       putchar(c);

fclose(fl);
}
```

Continuing with

David Lewis
&
Bob Hardin

Can you imagine the results of an industry standard being established in the software business? A standard that would set portable languages such as 'C' and 'SCULPTOR' the languages of choice in every new system that comes out. It wouldn't be a revolution but an end to one. Isn't it time the giants shook hands and agreed on something? Well if the Goliaths and the Trojan Horses won't find a way to relate I believe the time has come where the small guys (remember David) can bridge the gap between them. Both of these languages have taken what could be considered a GIANT step in that direction.

From my point of view as a user of the language, things seem quite obvious in my descriptions. It is hard to put myself in your shoes and see if it is still so obvious. I am depending on your feedback to help develop the direction of my pseudo-tutorial in future editions.

I received a letter from Cliff Rushing, asking whether SCULPTOR is command driven, menu driven, or fill-in-the-blank screens. First of all I want to point out that SCULPTOR is a programming language and the package comes with a number of support utilities that allow this powerful fourth generation language to do many things. As a matter of definition SCULPTOR is none of the things you mentioned. However, in using SCULPTOR you can develop programs that will do everything you asked.

The SCULPTOR language requires a specific series of events. First the data files must be described and created. Second the source program must be written either by the programmer or by one of the automatic program writing utilities supplied (more on this later). The source must then be compiled by the compiler into an object code. Execution of the object code is then handled by the interpreter.

In using the language the developers have supplied many utilities that make things very easy for the beginning user but still allows the full power of this system for advanced users. SCULPTOR works in 2 distinct but related ways. The first could be called automatic program development and the second a programmer development system.

I. Automatic Program Development

```
        (This is the SCULPTOR main menu)
     SCULPTOR DEVELOPMENT SYSTEM - MAIN MENU

 0 - FINISH - Exit from the SCULPTOR System
 1 - DESCRIBE - Create or edit a file descriptor
 2 - CRFILE - Create a blank data file
 3 - EDFORM - Input & Amend a Screen Form Program
 4 - COMFORM - Compile a Screen Form program
 5 - EDREP - Input & Amend a Report program
 6 - COMREP - Compile a Report program
 7 - RUNFORM - Run a Screen Form program
 8 - RUNSREP - Run a Report program (on the screen)
 9 - RUNPREP - Run a Report program (on the printer)
10 - QUERY - Run the Enquiry system
11 - AUTOFORM - Automatic Screen Form Program generation
12 - AUTOREP - Automatic Report Program generation
13 - DIR - Directory Listing

     Which option do you require?
```

The easiest way to use SCULPTOR is to allow the language and utilities do as much of the work as possible. SCULPTOR comes with a main menu as seen above for access to all the SCULPTOR programs and utilities. Select number 1 and you can create file descriptions. Option number 2 then makes them into empty keyed files. With menu selection number 11, and the name of the file, the utility will write a complete update program with add, find, next, amend, delete, and exit functions. The system then compiles this program and with selection 7 the program is executed. Automatic Program Development lets you command the menu to produce and run a fill-in-the-blank screen-form program. SCULPTOR can do some quite amazing things considering the power both of the language and the utilities included.

For a great many of the uses this application will work very well. Let's demonstrate a customer file maintenance program: Select option 1 then enter 'customer' as file name. We shall describe the fields as 'cusno' the main key field with 'cname', 'catreet', 'ccity', 'cstate', and 'czip' as data fields. The description will include each field name, a title or heading for the field, field type (alpha, money, date, integer, or real), size (in bytes), format (upper, lower, or numeric print positions), and an optional validation list to prevent unwanted data from being entered. The file is saved and menu returns.

```
  (This is the output from the describe program)
Descriptors for customer

List,Change,Delete,Insert,Abandon,Save,Help: L
KEY FIELDS
  1:cusno,Customer #,a6,

DATA FIELDS
  2:cname,Customer Name,a28,
  3:catreet,Street,a28,
  4:ccity,City,a16,
  5:cstate,State,a2,u
  6:czip,Zip,a5,

List,Change,Delete,Insert,Abandon,Save,Help: A
 Abandon (y/n)? y
```

Option 2 is then selected and customer again is typed in at the prompt. This creates a new, empty data and key file. Option 11 is then selected, customer is typed at the prompt and a program is written by the "sg" utility to insert, find, next, match, amend, delete, and exit. Option 7 is then selected, customer is entered and the program will be run.

```
(This is the screen form produced)
"CUSTOMER" FILE MAINTENANCE
                                    Today's date [        ]

          Customer # [      ]

  Customer Name [                          ]
        Street [                           ]
          City [                ]
         State [  ]
           Zip [     ]

i=insert   f=find   n=next   m=match   a=amend   d=delete   e=exit
```

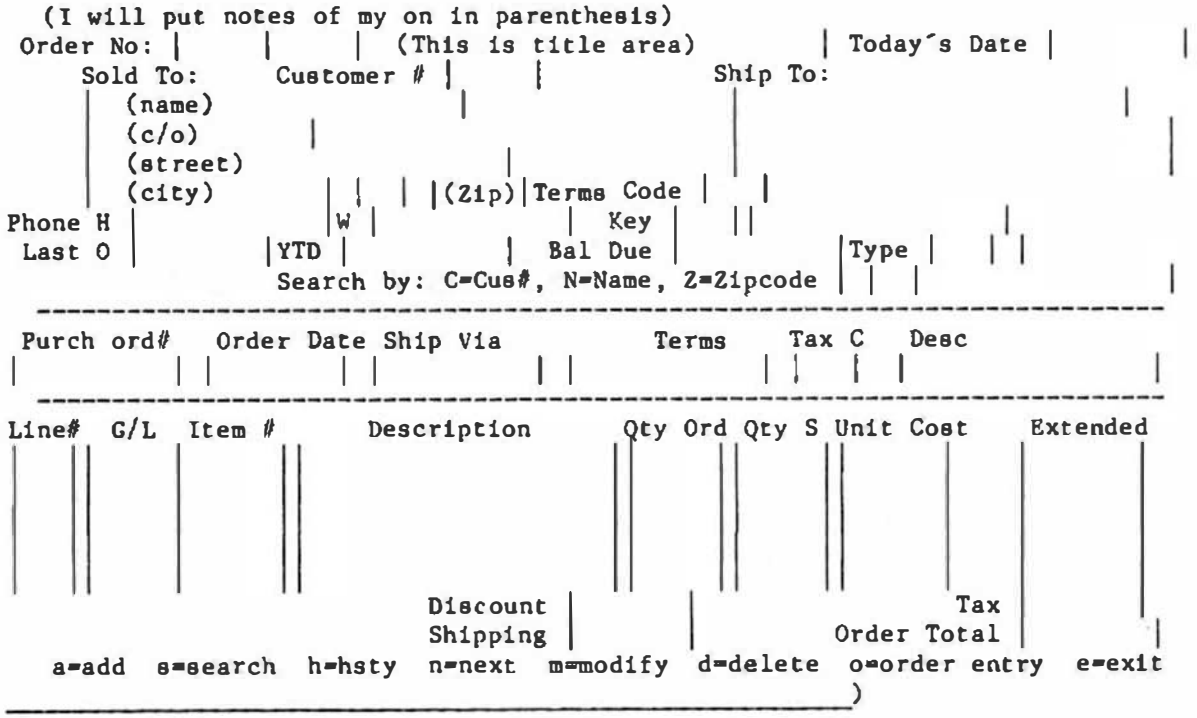
II. Manual Program Development

Now that we have skimmed over the automatic program development portion let's follow each type of program through the process of manually writing and executing. We will step through a screen form program and then a report program. Everything in quote ' marks are commands as typed on the system. We shall duplicate the customer program above. Creating the customer file is achieved by 'deacribe cuatomer' with field entries as previously indicated. Next we write a program using either the supplied editor or another editor of our choosing such as 'edit cuatomer.f' the .f means that it will be a screen program. In thia program we list the fields with the column and row on the screen we wish for the form to be displayed, followed by commands to add data, find data, amend data, delete data, and whatever else we want including exit. Thia file is then compiled by 'cf customer' the compiler will add the .f extension and if there are no errora found a file named cuatomer.g will be produced. This is then executed by 'sage customer' which actually runa the program. Here we enter data into the file, check it, correct it, etc. listed below is the program that is created by tbe 'ag' automatic program development utility that waa uaed in the firat example. With SCULPTOR even more detailed programs can be written manually.

(Thia ia the actual program produced by 'sg')

```
"CUSTOMER" FILE MAINTENANCE

!file cuatomer cuatomer

!temp date,,d4
+date,Today's date,2,70
+cuano,,4,43
+cname,,6,34
+cstreet,,7,34
+ccity,,8,34
+cstate,,9,34
+czip,,10,34

        display date
END\
        end

*i=inaert
        clear : display date
i1\
        message "Uae BACKSPACE to finish inserting"
        input cuano bs = i4
        read cuatomer nsr = i2
        gosub DISPLAY
        error "Already recorded"
        end
i2\
        input cname-czip ba = i1 eoi = i3
i3\
        insert customer
        clear : display date
        goto i1
i4\
        clear : display date
        end

*f=find
        clear : display date
        input cuano bs = END
        find customer
        gosub DISPLAY
        end

*n=next
        next customer
        gosub DISPLAY
        end

*m=match
        match customer nar = m1
        gosub DISPLAY
        end
m1\
        error "No further matching records"
        end

*a=amend
        check customer
        goto a1
a0\
        input cuano bs = END
a1\
        input cname-czip bs = a0 eoi = a2
a2\
        prompt "All correct" no = a1
        write customer
        clear : display date
        message "Record amended"
        end

*d=delete
        check customer
        prompt "Are you sure" no = END
        delete customer
        clear : display date
        end

*e=exit
        exit

DISPLAY\
        display cuano-czip
        return
```

The procedure to produce a program for printed results instead of screen updating is called a report program, created by 'edit customer.r'. Here we enter the statements to print our data exactly as we want it along with any batch updating we may want to do. It is compiled by 'cr customer' and executed by 'sagerep customer (printer_parameter_file) >/dev/spr' the printer parameter file is required to tell sagerep how to handle the printer we want. The output ie redirected to the appropriate device (/dev/spr or any other device), if not then it will be displayed on the screen. As you can see, writing powerful software in SCULPTOR is easy and effective. The above examples have been quite simple and much more complicated programs can be done very easily. I have written a customer update, order entry program that is designed for use specially in a telephone environment where time ia of the essence. The program is designed to take care of all facets of customer file maintenance and order entry without having to switch back and forth between programs. Below you will see the screen form generated by the program (the source program is 10 pages long.



The 'menu' is another great example of the ease with which you can produce powerful results. Below is the actual text file used by the SCULPTOR 'menu' program to create the menu found in the section I. above. Similar files can be used to create easy effective menus for your own software package.

```
SCULPTOR DEVELOPMENT SYSTEM - MAIN MENU
0,FINISH - Exit from the SCULPTOR System
exit
1,DESCRIBE - Create or edit a file descriptor
describe %
Please type the SCULPTOR file name
2,CRFILE - Create a blank data file
newkf %
Which file ia to be created
3,EDFORM - Input & Amend a Screen Form Program
sage /usr/sculptor/edit %.f
Which program do you wish to edit
4,COMFORM - Compile a Screen Form program
cf %
Which program is to be compiled
5,EDREP - Input & Amend a Report program
sage /usr/sculptor/edit %.r
Which program do you wish to edit
6,COMREP - Compile a Report program
cr %
Which program is to be compiled
7,RUNFORM - Run a Screen Form program
sage %
Which program do you wish to run
8,RUNSREP - Run a Report program (on the screen)
sagerep % pvdu | more; pause
Which program do you wish to run
9,RUNPREP - Run a Report program (on the printer)
sagerep % |ppr
Which program do you wish to run
10,QUERY - Run the Enquiry system
sage /usr/sculptor/query
11,AUTOFORM - Automatic Screen Form Program generation
sg %;ed in %.f
Which SCULPTOR file do you want a program for
12,AUTOREP - Automatic Report Program generation
rg %;ed in %.r
Which SCULPTOR file do you want a program for
13,DIR - Directory Listing
dir %; pause
Which directory do you wish to list
```

If you want to create your own menu let me explain how the text file supplied for the main system menu works. The 'menu' program takes a text file as above and displays a menu with it. The first line is the title line. Numbered lines are option lines with the command lines following them. When the menu program encounters a '%' sign it displays the next line as a message prompt and awaits the input of an argument to replace the '%' sign. If you will refer back to the January issue you will find a description of all the programs in the above menu except the edit programs (edit, ed_in) which are programs written in SCULPTOR that can be used for creating and editing source programs. Of course, any good text editor could be used.

I received some literature from the developers of SCULPTOR about proposed enhancements (including an indexed manual) in fact I found it interesting that most of these enhancements in progress were mentioned in my initial review (January '86). I won't go into detail at this time but as described they should make a good thing even better. When they are available I shall go into more detail about them. Keep the letters coming!

. . .

Editor's Note: Many current UniFLEX and OS-9 commercial software programmers are using **SCULPTOR +.** At a recent SWTPC dealer meeting it was determined that over 70% of those present were writing most all their custom applications in Sculptor +.

S.E. MEDIA is USA distributor for Sculptor Plus. This means that you dealers & multi-station users should check with S.E. MEDIA before purchasing **Sculptor Plus.** Also if you have any question about that product you should give S.E. MEDIA a call. They support the product and offer rapid turn-around on orders and/or updates.

So if you are in the market for multiply copies of **Sculptor or are a paid subscriber of 68 MICRO JOURNAL,** then you qualify for a discount. Give S.E. MEDIA a call.

DMW

- - -

COMPACTA UNIBOARD

Dear Don

The 18 month delay between promising you this Uniboard review and fixes article and getting it to you was not due to the mail system! I trust the material may still be of interest to you and the readers.

I am trying to clean up the spooling at present. Ron Anderson's recent tutorials on the format of Flex files led to my resolving a problem with spooling from Stylo. In this case the text expansion done by Flex was ruining all my printer escape sequences which, in accordance with Murphy's law, were full of $09 characters. I have modified the Flex spooler to inhibit expansion and hope one day to change the PRINT command so I can optionally specify expansion or not when opening a new print queue. This also requires an option feature when writing text files to use either mode. Perhaps someone has already done this - if so I would be interested to hear from them.

Please pass on my everlasting gratitude, love and kisses or whatever, to Ron Anderson for his timely tutorial comments. My thanks also to you and all the crew and other contributors which combine to put more USEFUL information per millimetre in 68MJ than in any other computing magazine I know. (The others are half full of mundane space invader games!!)

Regards

a/Kingsley Burlinson

Editor's Note: Thanks Kingsley, it is never too late. In fact at the same time this is being run in 68 Micro Journal, Digital Research is offering a very special offer on the board and support chips.

We will also offer this on our reader service disk for those who do not want to type or assemble all the source.

I and thousands of readers thank you for your sharing.

DMW

- - -

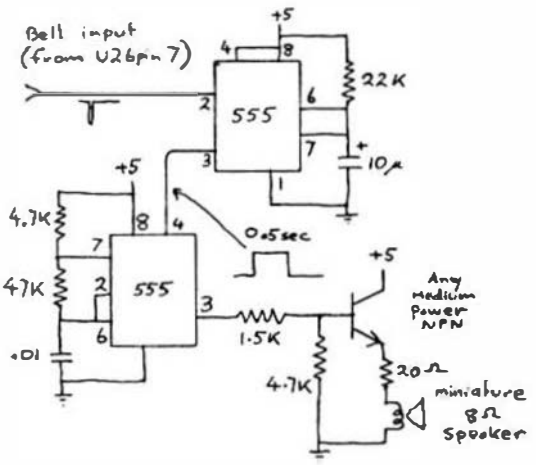
REVIEW OF THE COMPACTA UNIBOARD
K.Burlinson
P.O. Box 37134 Winnellie N.T. 5789 Australia

The Compacta Uniboard, as advertised in 68 Micro Journal, had all the features I had been wanting in a microcomputer. A single board carried everything from the disk controller to a video display driver, with 64K of RAM included. It follows the concept of the Z80 Big Board, also from Compacta, but uses the 6809 microprocessor (so of course, it has to be better!) and runs Flex or OS9. Being hardware oriented, I was quite happy to buy just the bare board and assemble the computer myself.

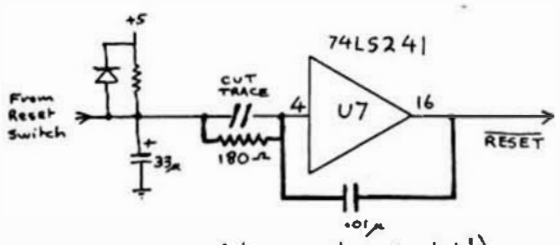
I purchased the board late in 1983 and completed building the system in mid 1984. (It took a long time to get here from USA) This review has been written over the ensuing 18 months and is already rather old. Another review has already been published in 68 and some of the bugs I found, along with some extra ones, have also been described by Sillanpaa in 68 MJ of Sept. 85. I only purchased the Flex system and cannot comment on how the board works with OS9.

The Hardware End

The board itself is well made and comes with 2 PAL chips and the monitor ROM, a wad of documentation and a (much photoreduced) circuit diagram. There are 2 repairs necessary to the printed circuit board and these are documented on a seperate loose sheet provided. The changes are minor and no real problem, but quite necessary. The only surprise was the use of 32 type 4116 dynamic RAMs instead of more modern chips. It is tedious soldering all those memory chips, but at least they do work reliably. The step by step assembly instructions provided are accurate and quite detailed so assembly is no problem as long as you take care with the fine soldering required. Once assembled, only a few adjustments are required. These are well explained in the documentation and easy to perform, but you will need a CRO and a voltmeter to do them.



Bell circuit - momentary pulse input
Suitable for Uniboard



(changes shown in bold)

Uniboard - change to reset circuit

My board worked the first time I powered it up and has been reliable over the 18 months since then. Clearly, the hardware is well designed. It uses a DMA controller for high speed data transfers (e.g. to the disk) and the 6845 video controller provides the dynamic RAM refresh. Memory accesses of the microprocessor and the on board memory mapped video are interleaved and there is absolutely no hash on the screen during display changes. You can use a terminal if you wish - that circuitry is included - but I have not used it with a terminal, nor have I yet used either the RS232 or 20ma current loop serial ports provided. Although a bell signal is decoded on the board, this output is not actually connected anywhere. To use it you need to run a wire to pin 7 of U26 (not shown on the circuit diagram) and provide your own bell circuitry. I built this circuitry into my keyboard and used a spare pin on the keyboard connector to wire it up. (Fig. 1)

I did experience a problem with the reset circuitry. The power on and manual reset operations usually resulted in a locked up system and only repeated operation of the reset button would eventually unlock the system. Inspection with the CRO showed that the reset line had several pulse edges in rapid succession rather than a single clean edge. The 6800 specifications specifically note that this is not allowed and although no reference is made to this in the 6809 data sheets, it seems it is still disallowed as cleaning up this pulse solved the problem. The problem is due to oscillation of U7, a 74LS241 buffer which is being used to square up a ramp input signal. This chip has an RC network and the reset button on it's input. It is preferable to use a schmitt trigger or a CMOS gate in such a circuit to avoid problems due to the current sourced from a TTL input. To solve the problem with minimal circuit board changes I provided some positive feedback around this gate to force a rapid and stable state transition.(Fig. 2) This solves the problem - but requires cutting some PCB traces and voids the warranty! If that worries you, try using only the capacitor, without the series resistor, as that would avoid having to cut the PCB traces. Or try using different 74LS24ls.

The documentation of the hardware is terse and lacks one or two items, but is sufficient. It is not well laid out and you do need to go through all the information carefully to learn how to set up some of the jumpers. One jumper, (P6), for the 20ma current loop, is not described at all in the documentation and there are no details of the pinouts for the printer connector. All the other connectors are adequately described. A bus expansion connector is provided and seems to be properly documented although some aspects of the use of this connector are mentioned only very briefly.

The Software End

The firmware supplied in the monitor ROM provides all the basic system functions, some simple but very useful commands (to read, write and test memory, etc.) and supports the inbuilt video console. It is quite comprehensive and very good in making the implementation of FLEX very simple. However it is not well documented, there being several errors and omissions in the notes. There are also 2 significant errors in the ROM, one of which was quite a disaster for me!

The main problem is that the disk drivers provided in the ROM do not support double sided disks, ... despite the documentation and source code claiming that they do. So if you only plan on using single sided drives or if you are content to configure your double sided drive(s) as two independant logical drives, then the ROM routines should work OK for you. Otherwise, you need to bypass several of the disk driver routines in ROM. This is easily done by modifying the Flex jump table to point to the repaired drivers loaded into the flex disk I/O area. The actual changes necessary to the drivers are quite minor and listed below - but you need to purchase the optional (but essential) monitor source code when you purchase your Uniboard.

(In the year and a half since I contacted Digital Research Computers and Compacta, I have heard from neither and I do not know whether a new ROM is now being supplied. My board was one of the early ones.)

The Uniboard supports 4 drives, either 8" or 5", but only 2 of either size are allowed. I use 2 8" double sided drives. If you plan to mix your disk sizes and also want to use double sided drives you will need to repair an additional routine, CHKRDY, which checks that a drive is ready and performs the selection of whether this drive is 8" or 5".

An additional software problem - not really an error - occurs in the video console code. (This will not affect those using a terminal). The console supports a subset of the VT52 terminal commands and works very well. The problem only occurs with reverse video. The hardware supports this and characters with bit 7 set are displayed in reverse video. Unfortunately the ROM routines regard any character with bit 7 set as unprintable and ignore them. To use reverse video you must bypass the ROM routines and write directly to the video ram yourself. A short routine can easily be added to your applications to allow use of the reverse video feature. This need merely check bit 7 of a character returning from the console and, if set, grab the cursor pointer, write the character directly to the video ram and then increment the cursor pointer.

A further software error is present in the PRINT.SYS file supplied on the source disk. This will only allow one file to be printed in a session, requiring a system reset before another file can be printed, and the print spooler will not work at all. The file provided is a direct copy of the example provided in the Flex manual and has not been modified to read the Printer Ready status byte at the address used in the ROM. The address of this byte should be changed to $EE17, the location that the monitor ROM uses for this function. The code below avoids this by using the PRINTSYS routines within the ROM.

The documentation of the software is, in places, misleading. For example, the stated required contents of the processor registers at entry into the write track routine are wrong. If you plan to use the ROM routines I suggest that you read the monitor source listing rather than rely on the documentation. Note also that when you are using the monitor commands they require 4 character data fields - even if only a 2 character value is being entered, such as the data byte value in the fill memory command. In this case the first two characters are used and the last 2 ignored (The same as with the Z80 big board).

Despite these failings, the ROM software does have some significant good points. All the hard work of getting Flex running has been done. Not only are the drivers already written but 3 monitor commands are provided to boot various species of Flex. Once your hardware is running you merely need to execute the L command and you are running Flex! (The general version from TSC) Note that you can do this before fixing the disk drivers - it merely restricts you to single sided disks until you make the repairs. The software supplied on the source disk completes all that is required to establish a normal Flex system, although the Uniboard documentation fails to explain this or exactly how to piece it together so you will need to read the Flex documentation carefully. This is not such a bad idea anyway. Having had to fight with the CPM documentation in the past it was great to see that Flex is very well documented.

Conclusion

Overall, the hardware on this board is well designed, comprehensive and reliable. The software has a few bugs and is not well documented but is workable. Just make sure you purchase the essential monitor source code on disk. The source listing is very well commented and between this and the documentation you have all the data you need to make repairs and alterations. The completed system has given me no problems in running all the normal flex software including Stylo, XDMS, Crasmb, Dynamite and Introl C. Most of these require no modifications to use the video console instead of a terminal. Stylograph is easily set up to use the console using the instructions provided. Even the reverse video feature for character modifications in Stylo can be used by incorporating some code to overcome the bug in the ROM. (see below)

I do not now regret purchasing this board, although I wasn't so positive during the disk driver debugging! Having only used a 6800 processor until this purchase I was planning on learning 6809 assembler. Having to sort out these problems to get the system running just made me learn 6809 code much faster! Using the fixes below and others previously published in 68MJ, you should not experience much trouble in getting a Uniboard running.

** Note: This is available in source with documentation (as above) from the 68 MICRO JOURNAL Reader Disk Service, see adv this issue.

DMW

\- - -

```
*****************************************************
*
* TERMINAL DRIVER ROUTINES TO USE STYLOGRAPH ON THE
* COMPACTA UNIBOARD USING VIDEO CONSOLE DISPLAY
*
* YOU NEED TO INSERT YOUR OWN IOBEG VALUE
* OBTAINED AS DESCRIBED IN THE STYLO MANUAL
*
* THIS ROUTINE DISPLAYS MODIFIED CHARACTERS IN REVERSE VIDEO
* CONSOLE DRIVER IN ROM DOES NOT WORK IN REVERSE DUE TO BUG
*
*     K.BURLINSON    DARWIN    1985
*
*****************************************************


*HARDWARE I/O ROUTINES BRACKETED BY
*LABELS "IOBEG" & "IOEND" SO THEY CAN
*BE CUSTOMIZED BY THE USER.

*SET BY USER
IOBEG EQU $XX -----SET IOBEG ------------ SET THIS FOR YOUR COPY

 ORG IOBEG
```

```
*CONSTANTS
PORT FDB $EF21 ACIA ADDRESS    UNIBOARD PIA INPUT ADDRESS

*    MODIFIED FOR INVERSE VIDEO FEATURE


PAMEJM FDB $FFFF HIGHEST RAM USED IF LOWER THAN MEMEND
SIMFLG FCB 0 NON-ZERO IF NO INTERRUPTS
PRTFLG FCB 1 ZERO IF OUTPUT THROUGH FLEX, ELSE PRTOUT
INTVEC FDB $3FC8 INTERRUPT VECTOR LOCATION


 ORG IOBEG+$10 SET BRANCH TABLE
*BRANCH TABLE

INTON BRA JINTON
INTOFF BRA JINTOFF
PINIT BRA JPINIT
PCHECK BRA JPCHECK
GETCH BRA JGETCH

* EQUATES USED FOR COMPACTA UNIBOARD

SCNI EQU $EE21 RAM ADDRESS OF SCREEN POINTERS
CURSOR SET 4 OFFSET OF CURSOR POINTER
ESCAPE SET 11 CURSOR OFFSET IN SCREEN TABLE
DIRCSR SET 10

PRTOUT EQU *


*OUTPUT CHARACTER THROUGH ACIA
 PSHS X,Y,D,U
 LDU $SCNI      GET SCREEN TABLE ADDRESS
 CMPA #$20      IS CHARACTER PRINTABLE
 BLT NOUT       JUMP IF NOT
 TST ESCAPE,U   IS ESCAPE SEQUENCE UNDERWAY
 BNE NOUT       JUMP IF SO
 TST DIRCSR,U   IS CURSOR ADDRESS IN PROGRESS
 BNE NOUT       SKIP IF SO
 TST INVFG      IS INVERSION REQUIRED
 BEQ NOUT       NO, SO SKIP
 ORA #$80       SET HIGH BIT FOR INVERSION
*
* THE CONSOLE WILL IGNORE THIS CHARACTER WITH THE HIGH BIT SET
* MUST CHECK IT UPON RETURN FROM CONSOLE AND DO OUR OWN
* OUTPUT IF REQUIRED
*
NOUT JSR [$FFBE] CONSOLE OUTPUT ENTRY POINT, CHARACTER IN A REG.
 TSTA           IS THE HIGH BIT SET
 BMI CINV       YES, SO GO WRITE INVERTED CHAR
 TST ESCFG      WAS PREV CHAR AN ESCAPE
 BEQ GOBAK      RETURN IF NOT
 CMPA #']       IS THIS THE INVERT COMMAND
 BNE UNDO       SKIP IF NOT
 INC INVFG      SET INVERT FLAG
 BRA GOBAK      AND RETURN
UNDO CMPA #'N   IS THIS THE CANCEL COMMAND
 BNE GOBAK      SKIP IF NOT
 CLR INVFG      CANCEL INVERSION
 BRA GOBAK      RETURN
CINV LDY CURSOR,U   THIS CHAR IS INVERTED, SO GET CURSOR
 STA 0,Y        OUTPUT CHAR DIRECT TO VIDEO MEMORY AREA
 STY CURSOR,U   INCREMENT AND UPDATE CONSOLE CURSOR
GOBAK LDA ESCAPE,U  GET MONITOR ESCAPE FLAG
 STA ESCFG      AND UPDATE LOCAL FLAG
 PULS X,Y,D,U,PC  AND RETURN


*GET A CHARCTER
JGETCH JMP [$FF88] MONITOR GETCHARACTER ROUTINE

*CHECK FOR CHARACTER AT INPUT
JPCHECK JMP [$FFBC] MONITOR CHECK FOR CHARACTER ROUTINE

*ALLOW ACIA TO GENERATE INTERRUPTS
JINTON RTS NOT USED - DISABLE
```

```
*DUMMY DISABLE INTERRUPTS
JINTOFF RTS NOT USED - DISABLE

*INITIALIZE PORT
JPINIT RTS UNREQUIRED - DISABLE
INVFG FCB 0 FLAG TO INDICATE REVERSE VIDEO IS REQUIRED IF <>0
ESCFG FCB 0 FLAG IF LAST CHARACTER WAS AN ESCAPE

*ADD A FEW BYTES FOR USER REVISIONS
 PPT 0
 FDB 0

* MODIFIED PRINTSYS   K.BURLINSON   DARWIN  1985

*  PRINT.SYS USING ROM ROUTINES FOR UNIBOARD
*  COMPACTA SUPPLIED PRINTSYS.TXT IS WRONG
*  IT DOESN'T USE THE CORRECT PRINTER READY FLAG
*  THIS ROUTINE USES THE INBUILT ROM DRIVERS


  ORG $CCC0
 JMP [$FFEA] PRINTER INITIALIZATION ROUTINE

 ORG $CCC8
 JMP [$FFEC] PRINTER READY ROUTINE

 ORG $CCE4
 JMP [$FFEE]  PRINT CHARACTER ROUTINE

 END $CCC0

*     DISK IO ROUTINES FROM COMPACTA MONITOR
*     MODIFIED TO INCLUDE MISSING SIDE SELECT FUNCTION
*
*  K.Burlinson   Darwin   June 84/Dec 85
*
* Almost all this code is extracted verbatim from the Compacta
* source code. Only 7 new lines of code are added.
* Because these routines refer to each other it is
* necessary to reassemble them as a group. The remaining
* disk routines are used in the ROM.
*
* Purpose of the changes is to add a side select capability
* to the SEEK TRACK routine and also to force a switch to
* side 0 when a RESTORE TO TRACK 0 is executed.
*
* The assembled code is merely appended to Flex and resides
* in the normal Flex Disk I/O area at $DE00. The Disk
* driver table at $DE08 is modified so that these routines
* in RAM are used instead of the erroneous ROM versions.
*
*****************************************************************
*
* Monitor for Compacta's Uniboard-89E
* Copyright 1982 Compacta Incorporated
*
* P.O. Box 7484
* Newark, Delaware 19711
*

   ORG  $EE0C

* RAM LOCATIONS USED BY THE MONITOR ROUTINES

PIA EQU $E804 PIA CONTROL PORT IMAGE

*  IO DEFINITION

IOBASE EQU $EF00 BASE ADDRESS FOR ON BOARD I/O
FDCBAS EQU IOBASE+40 BASE ADDRESS FOR 1793 FDC
CNTBAS EQU IOBASE+72  BASE ADDRESS FOR CONTROL PORT


* DISK CONTROLLER DEFINITIONS

COMREG EQU FDCBAS COMMAND/STATUS REGISTER
TRKREG EQU FDCBAS+1 TRACK REGISTER
SECREG EQU FDCBAS+2 SECTOR REGISTER
DATREG EQU FDCBAS+3 DATA REGISTER

*
*
* DISK PARAMETERS

RDCMND EQU $8C READ COMMAND
WTCMND EQU $AC WRITE COMMAND
RSCMND EQU $08 RESTORE COMMAND
SKCMND EQU $1B SEEK COMMAND
*
* PORT EQUATES:
*
PI    EQU    CNTBAS   AUXILIARY CONTROL PORT
*
* PORT 1 DEFINITIONS:

SIDE   EQU   4        SIDE SELECT BIT
DDEN   EQU   8        DENSITY SELECT BIT (0=SINGLE)
MDDEN  EQU   $F7     COMPLEMENT OF DDEN

*
* DISK DRIVERS FOR FLEX


* EQUATES

RDY    EQU   $80      DRIVE READY
DRQ    EQU   2        DRQ BIT MASK
BUSY   EQU   1        BUSY BIT MASK
RDMSK  EQU   $1C      READ ERROR MASK
VERMSK EQU   $10      VERIFY ERROR MASK
WTMSK  EQU   $5C      WRITE ERROR MASK
PRCNT  EQU   $CC34
SMARD EQU $05
SMAWR EQU $04
SECDD   EQU   26    SECTORS/SIDE, DOUBLE DENSITY
SECSD   EQU   15    SECTORS/SIDE, SINGLE DENSITY
CSIDE   EQU   $FB   SIDE CONTROL BIT MASK

*****************************************************************
* THE FOLLOWING BRANCHES ARE DIRECT TO ADDRESSES IN ROM
* NOT VECTORED - NECESSITATED BY SPLITTING THE DRIVERS
* BETWEEN ROM AND RAM
*****************************************************************

DISKGO EQU $F597 DISKGO ROUTINE IN ROM
WCR EQU $F5B6 WRITE COMMAND ROUTINE IN ROM
SWIDEN EQU $F4C4   SWITCH DENSITY ROUTINE IN ROM
DEL28 EQU $F579    DELAY ROUTINE IN ROM  28MS??
FNDDEN EQU $F552    FIND CURRENT DENSITY ROUTINE IN ROM
FNDTRK EQU $F54A    FIND CURRENT TRACK ROUTINE IN ROM
DRV EQU $F4ED    CHECK DRIVE READY ROUTINE IN ROM

*****************************************************************
*     MODIFIED JUMP TABLE WITH RAM AND ROM JUMPS
*****************************************************************
*
  ORG $DE00

 JMP >READ     POINT THIS JUMP TO THE NEW RAM CODE
 JMP >WRITE    POINT AT THE DISK WRITE CODE IN RAM
 JMP >$DE2F    VERIFY LAST SECTOR WRITTEN - ROM
 JMP >RST      RESTORE TO TRACK ZERO  RAM
 JMP >$DE3B    CHECK DRIVE READY  ROM
 JMP >$DE37     CHECK DRIVE READY  ROM   SAME AS ABOVE
 JMP >$DE3F     QUICK DRIVE CHECK   ROM
 JMP >$DE1F    INIT AND WARM DRIVER INIT   ROM
 JMP >$DE1E    WARM DRIVER INIT    ROM
 JMP >SEEK     SEEK TRACK AND SELECT SIDE  MODIFIED RAM CODE
 RTS
 JMP [$FFDC]   INIT AND WARM DRIVER INIT ROM POINTER
 FDB 0
 FDB 0
 FDB 0
 FDB 0
```

```
FDB 0
FDB 0
JMP [$FFD2]    VERIFY SECTOR WRITE ROM POINTER
FDB 0
FDB 0
JMP [$FFD8]    CHECK DRIVE READY ROM POINTER
JMP [$FFD6]    CHECK DRIVE  SAVE AS ABOVE
JMP [$FFD4]    QUICK DRIVE CHECK ROM POINTER

* READ ONE SECTOR THIS ROUTINE UNCHANGED***************
* THIS ROUTINE READS THE SPECIFIED SECTOR INTO MEMORY AT THE
* SPECIFIED ADDRESS. A SEEK IS PERFORMED. A SECTOR IS 256
* BYTES LONG.
* ENTRY: (X) = ADDRESS IN MEMORY WHERE SECTOR IS TO BE PLACED
*        (A) = TRACK NUMBER
*        (B) = SECTOR NUMBER
* EXIT:  (X) = DESTROYED
*        (A) = DESTROYED
*        (B) = ERROR CONDITION
*        (Z) = 1 IF NO ERROR, 0 IF AN ERROR

READ   BSR   SEEK      SEEK TO TRK
       BNE   READ6     IF SEEK ERROR GO HANDLE IT
       LDA   #RDCMND   LOAD FDC COMMAND
         LDB   #DMARD    AND DMA COMMAND
       LDY   #256      AND BYTE COUNT
       LBSR  DISKGO    START FDC AND DMA AND WAIT COMPLETION
READ6  BITB  #$1B      SECTOR OR TRACK NOT FOUND?
       BEQ   READ8     SKIP IF OTHER ERROR
       LBSR  SWIDEN    SWITCH DENSITY
READ8  BITB  #RDMSK    MASK ERRORS
       RTS


* WRITE ONE SECTOR   ** THIS ROUTINE UNMODIFIED **********
* THIS ROUTINE WRITES THE SPECIFIED MEMORY BUFFER AREA TO THE
* SPECIFIED SECTOR. A SEEK OPERATION IS PERFORMED. A SECTOR IS
* 256 BYTES LONG
* ENTRY: (X) = ADDRESS OF MEMORY BUFFER
*        (A) = TRACK NUMBER
*        (B) = SECTOR NUMBER
* EXIT:  (X) = MAY BE DESTROYED
*        (A) = MAY BE DESTROYED
*        (B) = ERROR CONDITION
*        (Z) = 1 IF NO ERROR, 0 IF ERROR

WRITE  PSHS  X         SAVE POINTER
       BSR   SEEK      SEEK IF NECESSARY
       BNE   WRITE6    IF ERROR
       LDA   #WTCMND   LOAD FDC COMMAND
       LDB   #DMAWR    AND DMA COMMAND
         LDY   #256      AND BYTE COUNT
       LBSR  DISKGO
WRITE6 BITB  #$1B      SECTOR OR TRACK NOT FOUND?
       BEQ   WRITEB    SKIP IF OTHER ERROR
       LBSR  SWIDEN    ELSE SWITCH DENSITY
WRITEB BITB  #WTMSK
       PULS  X,PC      restore pointer and exit

**********************************************************
* THIS ROUTINE MODIFIED TO SELECT DISK SIDE
**********************************************************
*SEEK THE SPECIFIED TRACK
* THIS ROUTINE SEEKS TO THE TRACK SPECIFIED. THE CORRECT SIDE
* size and density are selected before the seek.
* ENTRY: (A) = TRACK NUMBER
*        (B) = SECTOR NUMBER
* EXIT:  (A) = MAY BE DESTROYED
*        (X) = MUST BE PRESERVED
*        (B) = ERROR CONDITION
*        (Z) = 1 IF NO ERROR, 0 IF ERROR
SEEK   PSHS  A,X       SAVE SOME REGISTERS
       PSHS  B         SAVE SECTOR NUMBER
       STB   SECREG    SET SECTOR
       LBSR  SEL20     POINT TO CURRENT DRIVE DENSITY
       LDA   PIA      ASSUME DOUBLE DENSITY
       ANDA  #NDDEN
```

SOUTH EAST MEDIA

New Software Additions

TEXT EDITORS

CEDRIC – A screen oriented TEXT EDITOR with availability of "HELP" aid. Macro definitions, configurable "permanent definable **MACROS**" – all standard features and the fastest "global" functions in the west. A simple, automatic terminal config program makes this a real "no hassel" product. Only 6K in size, leaving the average system over 165 sectors for text buffer – appx. 14,000 plus of free memory! Extra fine for programming as well as text.

Regular $129.95

*** SPECIAL INTRODUCTION OFFER *** **FLEX $69.95**

- - - - - - - - - - -

PAT – A full feature screen oriented TEXT EDITOR with all the best of "PIE (tm)". For those who swore by and loved only PIE, this is for you! All PIE features and much more! Too many features to list. And if you don't like these, change or add your own. PL-9 source furnished. "C" source available soon. Easily configured to your CRT, with special config section.

Regular FLEX $129.50

*** SPECIAL INTRODUCTION OFFER *** **FLEX $79.95**

SPECIAL PAT/JUST COMBO

PAT & JUST (w/source) **FLEX $99.95**

Note: **JUST** in "C" source available for OS-9

**** See JUST advertising – S.E. MEDIA Catalog – this issue**

PROGRAMMERS & USERS TOOLS

SOLVE – OS-9 Levels I and II only. A Symbolic Object/Logic Verification & Examine debugger. Including inline debugging, disassemble and assemble. **SOLVE IS THE MOST COMPLETE DEBUGGER we have seen for the 6809 OS-9 series!** SOLVE does it all! With a rich selection of monitor, assembler, disassembler, environmental, execution and other miscellaneous commands, **SOLVE** is the **MOST POWERFUL** tool-kit item you can own! Yet, **SOLVE** is simple to use! With complete documentation, a snap! Everyone who has ordered this package has raved! See review – 68 Micro Journal – December 1985. No "blind" debugging here, full screen displays, rich and complete in information presented. Since review in 68 Micro Journal, this is our fastest ever!

Levels I & II only – OS-9 Regular $149.95

*** SPECIAL INTRODUCTION OFFER * $69.95**

**** NOTE:** Please note the special discounts (limited time) of assorted software in the S.E. MEDIA catalog in this and other issues.

Also please note the new policy on documentation of some S.E. MEDIA owned or licensed software products offered in their catalog, and repeated below.

Most all prgrams have the documentation in text disk file format. If you can print it out on your printer, the price is as shown in the catalog for the software. If you want us to print it out, please add $25.00. This is our average reproduction, short run cost. On most all items, the savings is well worth your doing the printing. However, this is only done to help save you hard earned dollars. All other software has vendor furnished documentation included in the price. Another – "your choice" S.E. MEDIA feature!

K-BASIC updates are now available. If you purchased K-BASIC prior to July 1, 1985 and wish to have your K-BASIC updated, please send $35 enclosed with your master disk to Southeast Media.

K-BASIC under OS-9 and FLEX will now compile TSC BASIC, XBASIC, and XPC Source Code Files

Telex 5106006630
(615)842-4600
SOUTH EAST MEDIA
5900 Cassandra Smith Rd.
Hixson, TN 37343
for information
call (615) 842-4601

CoCo OS-9™ FLEX™
SOFTWARE

K-BASIC now makes the multitude of TSC MBASIC Software available for use under OS-9. Transfer your favorite BASIC Programs to OS-9, compile them, Assemble them, and BINGO -- usable, multi-precision, familiar Software is running under your favorite Operating System!

SAVE $100.00

K-BASIC (OS-9 or FLEX), including the Assembler
!!! Special !!! ~~$199.00~~ $ 99.49

SCULPTOR

Sculptor for 68020 OS-9 and UniFLEX $995

Microprocessor Developments Ltd.'s Commercial Application Generator Program provides a **FAST** Commercial Application Development tool unavailable to the OS-9 and UniFLEX User before. Develop any Commercial Application in 20% of the normal required time; gain easy updating or customizing. PLUS, the Application can also be run on MS-DOS and Unix machines! **Sculptor** handles input validation, complex calculations, and exception conditions as well as the normal collecting, displaying, reporting, and updating information in an orderly fashion. Key fields to 160 bytes; unlimited record size; file size should be held to 17 million records. Utilizes ISAM File Structure and B-tree Key files for rapid access. Input and Output communication with other programs and files plus a library of ISAM routines for use with C Programs. Run-time included w/ the Development package; a compiled Application only needs a Run-time License. Additional charge for Networked Units. Prices for **Development Package/Run-time**. Discounts available for purchases of 5 or more Run-time Packages.

OS-9 / UniFLEX --			68000 UniFLEX --			MS DOS --	
IBM PC Zenix --	$995.00/$175.00		Altos Zenix --	$1595.00/$265.00		--	$595.00/$115.00
MS DOS Network --	*	**	UNIX --	*	**	PC DOS --	* **

* Full Developement Package ** Run Time Package Only Full OEM and Dealer Discounts Available!

!!! Special Buy Out !!! **SPECIAL** - Limited Quantity !!! Special Buy Out !!!

6809 FLEX SOFTWARE

TSC Flex Utilities	was $75.00	Now only $50.00
TSC Text Processor	was $75.00	Now only $50.00
TSC Flex Precompiler	was $50.00	Now only $35.00
TSC Flex Basic	was $75.00	Now only $50.00
TSC Flex Diagnostic	was $75.00	Now only $50.00
TSC Text Processor	was $75.00	Now only $50.00
TSC Assembler	was $50.00	Now only $35.00
TSC Precompiler	was $50.00	Now only $35.00
TSC Editor	was $50.00	Now only $35.00
TSC Utilities	was $75.00	Now only $50.00

DISKS ONLY

TSC Extended Precompiler	was $50.00	Now only $35.00
TSC Basic Flex	was $75.00	Now only $50.00
TSC Diagnostics	was $75.00	Now only $50.00
TSC Utilities	was $75.00	Now only $50.00

MANUALS ONLY

TSC Basic Precompiler	was $25.00	Now only $20.00
TSC Text Editor	was $25.00	Now only $20.00
TSC Mnemonic Assembler	was $25.00	Now only $20.00

DISKS ONLY - 6800 Software

TSC Editor	was $50.00	Now only $35.00
TSC Utilities	was $100.00	Now only $70.00
TSC Assemblers	was $50.00	Now only $35.00

MANUALS ONLY - 6800 Software

TSC Diagnostics	was $25.00	Now only $20.00
TSC Debug	was $25.00	Now only $20.00
TSC Text Processor	was $25.00	Now only $20.00

!!! Please Specify Your Operating System & Disk Size !!!

PROGRAMMING

ASSEMBLERS

ASTRUC09 from **Southeast Media** -- A "Structured Assembler for the 6809" which requires the **TSC Macro Assembler**. F, CCF - **$99.95**

Macro Assembler for **TSC** -- The FLEX **STANDARD** Assembler.
Special -- CCF **$35.00**; F **$50.00**

OSM Extended 6809 Macro Assembler from **Lloyd I/O**. -- Provides local labels, Motorola S-records, and Intel Hex records; XREF. Generate OS-9 Memory modules under FLEX. FLEX, CCF, OS-9 **$99.00**

Relocating Assembler w/Linking Loader from TSC. -- Use with many of the C and Pascal Compilers. F,CCF **$150.00**

MACE, by Graham Trott from **Windrush Micro Systems** -- Co-Resident Editor and Assembler; fast interactive A.L. Programming for small to medium-sized Programs. F,CCF - $75.00

MACE -- MACE w/ Cross Assembler for 6800/1/2/3/8 F,CCF - **$98.00**

TRUE CROSS ASSEMBLERS from **Computer Systems Consultants** -- Supports 1802/5, Z-80, 6800/1/2/3/8/11/HC11, 6804, 6805/HC05/146805, 6809/00/01, 6502 family, 8080/5, 8020/1/2/35/C35/39/40/48/C48/49/C49/50/8748/49, 8031/51/8751, and 68000 Systems. Assembler and Listing formats same as target CPU's format. Produces machine independent Motorola S-Text.
FLEX, CCF, OS-9, UniFLEX each - **$50.00**
any 3 - **$100.00**
the complete set w/ C Source (except the 68000 Source) - **$200.00**

XASM Cross Assemblers for **FLEX** from **Compusense Ltd.** -- This set of 6800/1/2/3/5/8, 6301, 6502, 8080/5, and Z80 Cross Assemblers uses the familiar **TSC Macro Assembler** Command Line and Soruce Code format. Assembler options, etc., in providing code for the target CPU's. Complete set, FLEX only - **$150.00**

CRASMB from **Lloyd I/O** -- 8-Bit Macro Cross Assembler with same features as OSM; cross-assemble to 6800/1/2/3/4/5/8/9/11, 6502, 1802, 8048 Sers, 80/85, Z-8, Z-80, TMS-7000 sers. Supports the target chip's standard mnemonics and addressing modes.
FLEX, CCF, OS-9 Full package -- **$399.00**

CRASMB 16.32 from **Lloyd I/O** -- Cross Assembler for the **68000**.
FLEX, CCF, OS-9 **$249.00**

DISASSEMBLERS

SUPER SLEUTH from **Computer Systems Consultants** -- Interactive Disassembler; extremely POWERFUL! Disk File Binary/ASCII Examine/Change, Absolute or FULL Disassembly. XREF Generator, Label "Name Changer", and Files of "Standard Label Names" for different Operating Systems

Color Computer	SS-50 Bus (all w/ A.L. Source)
CCO (32K Req'd) Obj. Only **$49.00**	F, **$99.00**
CCF, Obj. Only **$50.00**	U, **$100.00**
CCF, w/Source **$99.00**	O, **$101.00**
CCO, Obj. Only **$50.00**.	

DYNAMITE + from **Computer Systems Center** -- Excellent standard "Batch Mode" Disassembler. Includes XREF Generator and "Standard Label" Files. Special OS-9 options w/ OS-9 Version.

CCF, Obj. Only **$100.00**	CCO, Obj. Only **$ 99.95**
F, " " **$100.00**	O, " " **$190.00**
	U, " " **$300.00**

PROGRAMMING LANGUAGES

PL/9 from **Windrush Micro Systems** -- By Graham Trott. A combination Editor/Compiler/Debugger. Direct source-to-object compilation delivering fast, compact, re-entrant, ROM-able, PIC. 8 & 16-bit Integers & 6-digit Real numbers for all real-world problems. Direct control over ALL System resources, including interrupts. Comprehensive library support; simple Machine Code interface; step-by-step tracer for instant debugging. **300+** page Manual with tutorial guide. F, CCF - **$198.00**

WHIMSICAL from **Whimsical Developments** -- Now supports **Real Numbers**. "Structured Programming" WITHOUT losing the Speed and Control of Assembly Language! Single-pass Compiler features unified, user-defined I/O; produces **ROM**able Code; Procedures and Modules (including pre-compiled Modules); many "Types" up to 32 bit **Integers, 6-digit Real Numbers**, unlimited sized Arrays (vectors only); Interrupt handling; long Variable Names; Variable Initialization; Include directive; Conditional compiling; direct Code insertion; control of the Stack Pointer; etc. Run-Time subroutines inserted **as called** during compilation. Normally produces 10% less code than PL/9. F and CCF - **$195.00**

C Compiler from **Windrush Micro Systems** by James McCosh. Full C for FLEX except bit-fields, including an Assembler. Requires the TSC Relocating Assembler if user desires to implement his own Libraries. F and CCF - **$295.00**

C Compiler from **Introl** -- Full C except Doubles and Bit Fields, streamlined for the 6809. Reliable Compiler; FAST, efficient Code. More UNIX Compatible than most.
FLEX, CCF, OS-9 (Level II ONLY), U - **$575.00**

PASCAL Compiler from **Lucidata** -- ISO Based P-Code Compiler. Designed especially for Microcomputer Systems. Allows linkage to Assembler Code for maximum flexibility.
F and CCF 5" - **$99.95** F 8" **$99.95**

PASCAL Compiler from **Omegasoft** (now **Certified Software**) -- For the **PROFESSIONAL**; ISO Based, Native Code Compiler. Primarily for Real-Time and Process Control applications. Powerful; Flexible. Requires a "Motorola Compatible" Relo. Asmb. and Linking Loader.
F and CCF - **$425.00** One Year Maint. - **$100.00**

K-BASIC from **LLOYD I/O** -- A "Native Code" **BASIC Compiler** which is now **Fully TSC XBASIC compatible**. The compiler compiles to Assembly Language Source Code. A NEW, **streamlined**, Assembler is now included allowing the assembly of LARGE Compiled K-BASIC Programs. Conditional assembly reduces Run-time package.
FLEX, CCF, OS-9 Compiler **with** Assembler - **$199.00**

CRUNCH COBOL from **Compusense Ltd.** -- Supports large subset of ANSII Level 1 COBOL with many of the useful Level 2 features. Full FLEX File Structures, including Random Files and the ability to process Keyed Files. Segment and link large programs at runtime, or implemented as a set of overlays. The System requires 56K and CAN be run with a single Disk System.
FLEX, CCF; Normally $199.00
Special Introductory Price (while in effect) -- $99.95

FORTH from **Stearns Electronics** -- A CoCo **FORTH** Programming Language. **Tailored to the CoCo!** Supplied on Tape, transferable to disk. Written in FAST ML. Many CoCo functions (Graphics, Sound, etc.). Includes an Editor, Trace, etc. Provides CPU Carry Flag accessibility, Fast Task Multiplexing, Clean Interrupt Handling, etc. for the "Pro". Excellent "Learning" tool!
Color Computer **ONLY** - **$58.95**

VISA MasterCard

** SHIPPING **
Add 2% U.S.A.
(min. $2.50)
Add 5% Surface Foreign
10% Air Foreign

Availability Legends —
F = FLEX, CCF = Color Computer FLEX
O = OS-9, CCO = Color Computer OS-9
U = UniFLEX
CCD = Color Computer Disk
CCT = Color Computer Tape

*FLEX is a trademark of Technical Systems Consultants
**OS9 is a trademark of Microware

!!! Please Specify Your Operating System & Disk Size !!!

SOFTWARE DEVELOPMENT

BasicO9 XRef from **Southeast Media** -- This **BasicO9 Cross Reference** Utility is a BasicO9 Program which will produce a "pretty printed" listing with each line numbered, followed by a complete cross referenced listing of all variables, external procedures, and line numbers called. Also includes a **Program List Utility** which outputs a fast "pretty printed" listing with line numbers. Requires BasicO9 or RunB.

O & CCO obj. only -- $39.95; w/ Source - $79.95

Lucidata PASCAL UTILITIES (Requires LUCIDATA Pascal ver 3)

XREF -- produce a Cross Reference Listing of any text; oriented to Pascal Source.

INCLUDE -- Include other Files in a Source Text, including Binary; unlimited nesting capabilities.

PROFILER -- provides an Indented, Numbered, "Structogram" of a Pascal Source Text File; view the overall structure of large programs, program integrity, etc. Supplied in Pascal Source Code; requires compilation.

F, CCF --- **HASH Utility** 5" - $40.00, 8" - $50.00

DUB from **Southeast Media** -- A **UniFLEX** "basic" De-Compiler. Re-Create a Source Listing from UniFLEX **Compiled Basic Programs**. Works w/ ALL Versions of 6809 UniFLEX Basic. U - $219.95

FULL SCREEN FORMS DISPLAY from **Computer Systems Consultants** -- TSC Extended BASIC program supports any Serial Terminal with Cursor Control or Memory-Mapped Video Displays; substantially extends the capabilities of the Program Designer by providing a table-driven method of describing and using Full Screen Displays.

F and CCF, U - $25.00, w/ Source - $50.00

DISK UTILITIES

OS-9 VDisk from **Southeast Media** -- For **Level I** only. Use the **Extended Memory** capability of your SWTPC or Gimix CPU card (or similar format DAT) for FAST Program Compiles, CMD execution, high speed inter-process communications (without pipe buffers), etc. - SAVE that System Memory. Virtual Disk size is variable in 4K increments up to 960K. Some Assembly Required.

-- **Level I ONLY** -- OS-9 obj. only - $79.95; w/ Source - $149.95

O-F from **Southeast Media** -- Written in **BASICO9** (with Source), includes: **REFORMAT**, a BASICO9 Program that reformats a chosen amount of an **OS-9** disk to **FLEX** Format so it can be used normally by FLEX; and **FLEX**, a BASICO9 Program that does the actual **read** or **write** function to the special O-F Transfer Disk; user-friendly menu driven. Read the FLEX Directory, Delete FLEX Files, Copy both directions, etc. FLEX users use the special disk just like any other FLEX disk. **SPECIAL 60 DAY OFFER** O-$39.95

COPYMULT from **Southeast Media** -- Copy **LARGE** Disks to several **smaller disks**. FLEX utilities allow the backup of **ANY size disk** to any SMALLER size diskettes (Hard Disk to floppies, 8" to 5", etc.) by simply inserting diskettes as requested by **COPYMULT**. No fooling with directory deletions, etc. **COPYMULT.CMD** understands normal "copy" syntax and keeps up with files copied by maintaining directories for both host and receiving disk system. Also includes **BACKUP.CMD** to download any size "random" type file; **RESTORE.CMD** to restructure copied "random" files for copying, or recopying back to the host system; and **FREELINK.CMD** as a "bonus" utility that "relinks" the free chain of floppy or hard disk, eliminating fragmentation.

Completely documented Assembly Language Source files included.

ALL 4 Programs (FLEX, 8" or 5") $99.50

COPYCAT from **Lucidata** -- Pascal **NOT** required. Allows reading TSC Mini-FLEX, SSB DOS6B, and Digital Research CP/M Disks while operating under FLEX 1.0, FLEX 2.0, or FLEX 9.0 with 6800 or 6809 Systems. COPYCAT will not perform miracles, but, between the program and the manual, you stand a good chance of accomplishing a transfer. Also includes some Utilities to help out. Programs supplied in **Modular Source Code** (Assembly Language) to help solve unusual problems.

F and CCF 5" - $50.00 F 8" - $65.00

FLEX DISK UTILITIES from **Computer Systems Consultants** -- Eight (8) different Assembly Language (w/ Source Code) FLEX Utilities for every FLEX Users Toolbox: **Copy** a File with **CRC Errors**; **Test Disk** for errors; **Compare** two **Disks**; a fast **Disk Backup** Program; **Edit Disk Sectors**; **Linearize Free-Chain** on the Disk; print **Disk Identification**; and **Sort** and **Replace** the Disk Directory (in sorted order). -- **PLUS** -- Ten XBASIC Programs including: A **BASIC Resequencer** with **EXTRAS** over "RENUM" like check for missing label definitions, processes Disk to Disk instead of in Memory, etc. Other programs **Compare, Merge,** or **Generate Updates** between two BASIC Programs, check BASIC **Sequence Numbers**, compare two **Unsequenced** files, and 5 Programs for establishing a Master Directory of several Disks, and sorting, selecting, updating, and printing paginated listings of these files. A BASIC Cross-Reference Program, written in Assembly Language, which provides an X-Ref Listing of the Variables and Reserved Words in **TSC BASIC, XBASIC,** and **[illegible] BASIC** Programs.

ALL Utilities **Include Source** (either BASIC or A.L. Source Code).

F and CCF - $50.00

BASIC Utilities ONLY for **UniFLEX** -- $30.00

COMMUNICATIONS

CMODEM Telecommunications Program from **Computer Systems Consultants, Inc.** -- Menu-Driven; supports Dumb-Terminal Mode, Upload and Download in non-protocol mode, and the CP/M "Modem7" Christensen protocol mode to enable communication capabilities for almost any requirement. Written in "C".

FLEX, CCF, OS-9, UniFLEX; with complete Source - $100.00
without Source - $50.00

XDATA from **Southeast Media** -- A **COMMUNICATION** Package for the UniFLEX Operating System. Use with CP/M, Main Frames, other UniFLEX Systems, etc. Verifies Transmission using checksum or CRC; Re-Transmits bad blocks, etc. U - $299.99

GAME

RAPIER - 6809 **Chess** Program from **Southeast Media** -- Requires FLEX and Displays on Any Type Terminal. Features: Four levels of play. Swap side. Point scoring system. Two display boards. Change skill level. Solve Checkmate Problems in 1-2-3-4 moves. Make move and swap sides. Play white or black. This is one of the **strongest** CHESS programs running on any microcomputer, estimated USCF Rating 1600+ (better than most 'club' players at higher levels). F and CCF - $79.95

VISA MasterCard

** SHIPPING **
Add 2% U.S.A.
(min. $2.50)
Add 5% Surface Foreign
10% Air Foreign

"FLEX is a trademark of Technical Systems Consultants
"OS9 is a trademark of Microware

Availability Legends —
F = FLEX, CCF = Color Computer FLEX
O = OS-9, CCO = Color Computer OS-9
U = UniFLEX
CCD = Color Computer Disk
CCT = Color Computer Tape

!!! Please Specify Your Operating System & Disk Size !!!

BUSINESS

WORD PROCESSING

SCREDITOR III from Wiadrush Micro Systems -- Powerful Screen-Oriented Editor/Word Processor. Almost 50 different commands; over 300 pages of Documentation with Tutorial. Features Multi-Column display and editing, "decimal align" columns (AND add them up automatically), multiple keystroke macros, even/odd page headers and footers, imbedded printer control codes, all justifications, "help" support, store common command series on disk, etc. Use supplied "set-ups", or remap the keyboard to your needs. Except for proportional printing, this package will DO IT ALL!

6800 or 6809 FLEX or SSB DOS, OS-9 - $175.00

STYLO-GRAPH from Great Plains Computer Co. -- A full-screen oriented WORD PROCESSOR -- (uses the 51 x 24 Display Screens on CoCo FLEX/STAR-DOS, or PBJ Wordpak). Full screen display and editing; supports the Daisy Wheel proportional printers.
NEW PRICING —> CCF and CCO - $99.95, F or O - $179.95, U - $299.95

STYLO-SPELL from Great Plains Computer Co. -- Fast Computer Dictionary. Complements Stylograph.
NEW PRICING —> CCF and CCO - $69.95, F or O - $99.95, U - $149.95

STYLO-MERGE from Great Plains Computer Co. -- Merge Mailing List to "Form" Letters, Print multiple Files, etc., through Stylo.
NEW PRICING —> CCF and CCO - $59.95, F or O - $79.95, U - $129.95

JUST from Southeast Media -- Text Formatter developed by Ron Anderson; for Dot Matrix Printers, provides many unique features. Output "Formatted" Text to the Display. Use the FPRINT.CMD supplied for producing multiple copies of the "Formatted" Text on the Printer INCLUDING IMBEDDED PRINTER COMMANDS (very useful at other times also, and worth the price of the program by itself), "User Configurable" for adapting to other Printers (comes set up for Epson MX-80 with Graftrax); up to ten (10) imbedded "Printer Control Commands". Compensates for a "Double Width" printed line. Includes the normal line width, margin, indent, paragraph, space, vertical skip lines, page length, page numbering, centering, fill, justification, etc. Use with PAT or any other editor.

* Now supplied as a two disk set:

Disk #1: JUST2.CMD object file, JUST2.TXT PL9 source: FLEX - CC
Disk #2: JUSTSC object and source in C: FLEX - OS9 - CC

The JTSC and regular JUST C source are two separate programs. JTSC compiles to a version that expects TSC Word Processor type commands, (.pp .ep .ce etc.) Great for your older text files.

The C source compiles to a standard syntax JUST.CMD object file. Using JUST syntax (.p .u .y etc.) With all JUST functions plus several additional printer formatting functions. Reference the JUSTSC C source. For those wanting an excellent BUDGET PRICED word processor, with features some of the others have. This is it!

Disk (1) - PL9 FLEX Version only - F & CCF - $49.95
Disk Set (2) - F & CCF & OS9 (C version) - $69.95

SPELL8 "Computer Dictionary" from Southeast Media -- OVER 120,000 words! Look up a word from within your Editor or Word Processor (with the SPH.CMD Utility which operates in the FLEX UCS). Or check and update the Text after entry; ADD WORDS to the Dictionary, "Flag" questionable words in the Text, "View a word in context" before changing or ignoring, etc. SPELL8 first checks a "Common Word Dictionary", then the normal Dictionary, then a "Personal Word List", and finally, any "Special Word List" you may have specified. SPELL8 also allows the use of Small Disk Storage systems.

!! SPECIAL LIMITED TIME OFFER !! F and CCF - $99.95

DATA BASE - ACCOUNTING

XDMS from Westchester Applied Business Systems -- Powerful DBMS; M.L. program will work on a single sided 5" disk, yet is F-A-S-T. Supports Relational, Sequential, Hierarchical, and Random Access File Structures; has Virtual Memory capabilities for Giant Data Bases. XDMS Level I provides an "entry level" System for defining a Data Base, entering and changing the Data, and producing Reports. XDMS Level II adds the POWERFUL "GENERATE" facility with an English Language Command Structure for manipulating the Data to create new File Structures, Sort, Select, Calculate, etc. XDMS Level III adds special "Utilities" which provide additional ease in setting up a Data Base, such as copying old data into new Data Structures, changing System Parameters, etc.

XDMS System Manual - $24.95
XDMS Lvl I - F & CCF - $129.95
XDMS Lvl II - F & CCF - $199.95
XDMS Lvl III - F & CCF - $209.95

ACCOUNTING PACKAGES -- Great Plains Computer Co. and **Universal Data Research, Inc.** both have Data Base and Business Packages written in TSC XBASIC for FLEX, CoCo FLEX, and UniFLEX.

MISCELLANEOUS

TABULA RASA SPREADSHEET from Computer Systems Consultants -- TABULA RASA is similar to DESKTOP/PLAN; provides use of tabular computation schemes used for analysis of business, sales, and economic conditions. Menu-driven; extensive report-generation capabilities. Requires TSC's Extended BASIC.

F and CCF, U - $50.00, w/ Source - $100.00

DYNACALC from Computer Systems Center -- Electronic Spread Sheet for the 6809.

F, SPECIAL CCF and OS9 - $200.00, U - $395.00

FULL SCREEN INVENTORY/MRP from Computer Systems Consultants -- Use the Full Screen Inventory System/Materials Requirement Planning for maintaining inventories. Keeps item field file in alphabetical order for easier inquiry. Locate and/or print records matching partial or complete item, description, vendor, or attributes; find backorder or below stock levels. Print-outs in item or vendor order. MRP capability for the maintenance and analysis of Hierarchical assemblies of items in the inventory file. Requires TSC's Extended BASIC.

F and CCF, U - $50.00, w/ Source - $100.00

FULL SCREEN MAILING LIST from Computer Systems Consultants -- The Full Screen Mailing List System provides a means of maintaining simple mailing lists. Locate all records matching on partial or complete name, city, state, zip, or attributes for Listings or Labels, etc. Requires TSC's Extended BASIC.

F and CCF, U - $50.00, w/ Source - $100.00

DIET-TRAC Forecaster from Southeast Media -- An XBASIC program that plans a diet in terms of either calories and percentage of carbohydrates, proteins and fats (C P &%) or grams of Carbohydrate, Protein and Fat food exchanges of each of the six basic food groups (vegetable, bread, meat, skim milk, fruit and fat) for a specific individual. Sex, Age, Height, Present Weight, Frame Size, Activity Level and Basal Metabolic Rate for normal individual are taken into account. Ideal weight and sustaining calories for any weight of the above individual are calculated. Provides number of days and daily calendar after weight goal and calorie plan is determined.

F - $59.95, U - $89.95

** SHIPPING **
Add 2% U.S.A.
(min. $2.50)
Add 5% Surface Foreign
10% Air Foreign

VISA MasterCard

Availability Legends —
F = FLEX, CCF = Color Computer FLEX
O = OS-9, CCO = Color Computer OS-9
U = UniFLEX
CCD = Color Computer Disk
CCT = Color Computer Tape

*FLEX is a trademark of Technical Systems Consultants
™OS9 is a trademark of Microware

!!! Please Specify Your Operating System & Disk Size !!!

```
      TST   1,S       CHECK IF TRACK 0
      BEQ   SEEK10    BR IF TRK 0, WE'LL SET IT TO SINGLE DEN
       LDB   #SECDD    ASSUME DDENSE, GET SECTORS/SIDE
         TST   0,X     TEST DENSITY
      BNE   SEEK1     CONTINUE IF DOUBLE DENSITY
SEEK10 ORA  #DDEN     ELSE SWITCH TO SINGLE DENSITY


*****************************************************
* FOLLOWING LINES ARE NEW ADDITIONS TO SELECT SIDE
*****************************************************
      LDB   #SECSD    GET SECTORS/SIDE, SD
SEEK1 CMPB  0,S   CHECK SECTOR REQUESTED AGAINST MAX/SIDE
      BHS   SIDE0     STILL SIDE 0
      ANDA  #CSIDE     CHANGE TO SIDE 1
      BRA   WRIPT     GO OUTPUT TO CONTROL PORT
SIDE0 ORA  #SIDE      ENSURE SIDE 0
*****************************************************
* END OF CHANGES. NEW LABEL ADDED TO LINE BELOW
*****************************************************

WRIPT STA   PI
      STA   PIIM
      LEAS  1,S       CLEAN UP STACK
      LDA   0,S       GET DESIRED TRACK
      LBSR  FNDTRK    POINT TO CURRENT TRACK
      LDB   0,X       GET CURRENT TRACK
      STB   TRKREG    REFRESH FDC TRACK REGISTER
      LBSR  DEL20
      CMPA  0,X       CHECK IF ON THE RIGHT TRACK
      BEQ   SEEK4     IF SO, EXIT WITHOUT SEEKING
SEEK2 STA   DATREG    ELSE SET NEW TRACK
      LBSR  DEL20
      LDA   #SKCMND
      LBSR   WCR
      TFR   B,A       SAVE ERROR
      LBSR  FNDTRK    POINT TO CURRENT DRIVE TRACK STORE
      LDB   TRKREG    GET CURRENT TRACK
      STB   0,X       STORE
      TFR   A,B       RESTORE ERROR
      BITB  #$10
      LBSR  DEL20     DELAY
SEEK4 PULS  1,A,PC    restore registers and exit
*****************************************************
* EXTRA LINE ADDED TO SELECT SIDE ZERO WHEN RESTORED
*****************************************************
* RESTORE TO TRK 0

RST   PSHS  X         SAVE POINTER
       LBSR   DRV     SELECT DRIVE
      LDA   PIIN      GET PORT CONTROL BYTE

*****************************************************
* FOLLOWING LINE ADDED
*****************************************************
      ORA   #SIDE      FORCE TO SIDE 0

      STA   PI        WRITE TO LATCH
      STA   PIIM      UPDATE RAM COPY OF LATCH BYTE
      LDA   #RSCMND
      LBSR  WCR
      PSHS  B         SAVE ERROR
      LBSR  FNDTRK    POINT TO TRACK STORE FOR CURRENT DRIVE
      LDB   TRKREG    GET TRK # FROM FDC
      STB   0,X       STORE NEW TRACK # FOR THIS DRIVE
      PULS  B,X       RESTORE ERROR AND POINTER
      BITB  #$9B
       RTS

        END
```

Bit Bucket

8 DEC 1985
321-B POINCIANA PL
HONOLULU, HI 96818

Dear Mr. Williams,

Merry Christmas! and Happy New Year, too! As my contribution to the spirit of the season, I am sending you a copy of my new index to 68 Micro Journal. This is just my way of saying thanks for your support in the last year. I also want you to know how much I appreciate your magazine. You are providing a much needed service, and this is just my way of saying "Thanks a lot"!

This index is a standard Flex text file which has proved very useful to me. One of the main values of 68 MJ is the useful little tid-bits that are often included in letters, asides, and the Bit Bucket. It's very difficult to track these back down some months later. I've tried to develop a comprehensive index including all bits of info I felt might be useful later. It's invaluable for finding those patches to contributed software that appear some months later.

The index is a key word index. Each line starts with month, year and page number of an article or item, and usually includes the author's name. I've also tried to identify the item as article, letter, program, utility, etc. Then follows certain key words selected to characterize the topic(s) covered. I've attempted to stay within the 128 character limitation of the Flex line buffer.

You may then use Leo Taylor's FIND.CMD to locate a specific topic of interest. When FIND locates a match, it prints the entire line including the date and page. This makes the whole operation quite simple, and I didn't even need to write any software! FIND.CMD syntax:

+++FIND,<FILENAME>,<STRING>

prints all lines containing "string". Default extension is .TXT.

The only confusion seems to be with names that are hyphenated or slashed. To

simplify things, I have generally just deleted these extra characters.

Examples:
SS30 NOT SS-30
CPM NOT CP/M
OS9 NOT OS-9
MPS2 NOT MP-S2
CFM3 NOT CFM/3
IO NOT I/O
PL9 NOT PL/9

I've included text files for each individual year, although on my own system I append them all together. This is slower than searching a specific year, hut the speed is adequate to me. I hope you can find this to be of some use to you. For fun just "find" all the references to "WILLIAMS". Please feel free to duplicate this, or give away to anyone else you wish.

Sincerely,

John Current

Ed's Note; Thank You John! I used your index to locate and sort several different topics and I am happy to say that the results were fantastic. Your system of indexing all 7 years of articles and Bit Bucket items was very well done!

We here are so impressed with your indexing that we have made all 7 years available as Reader Service Disk #24. See 68' Micro Disks Ad on Page 62.

JAN 85 P7 ANDERSON FLEX USER NOTES RAPID SERIAL DATA TRANSFER SEND RECEIVE BINARY ASCII PL9 LISTING

JAN 85 P9 PASS C USER NOTES COMPILER DSICUSSION STRING HANDLING OKEEFE FUNCTION LIBRARY SCAN OPTIONS C LISTING

JAN 85 P11 LUCIDO 68000 USER NOTES UNIX KERNEL USER INTERFACE UTILITIES

JAN 85 P13 DIBBLE OS9 USER NOTES SEMINAR OS968K MACHINES JAPAN UNIFLEX VM

JAN 85 P15 REVIEW COMPACTA UNIBOARD 6809 SINGLE BOARD COMPUTER DIGITAL RESEARCH COMPUTERS OMA

JAN 85 P19 OPYRCHAL ARTICLE CRUNCH COBOL MODULES

JAN 85 P20 DEVELOPMENT TERMINAL PROGRAM CONT. 6800 ASSEMBLY LISTING

JAN 85 P22 SLAGHEKKE DO FLEX COMMAND FILE PROCESSOR 6809 ASSEMBLY LISTING

JAN 85 P29 TURNER REVIEW ELEKTRA SUPER FLOPPY CONTROLLER DISK SS30

JAN 85 P30 MILLS FLEX UTILITY PROMPTED FILE TRANSFER PARITY CHECKING MODEM 6809 ASSEMBLY LISTING

JAN 85 P41 REVIEW KBASIC COMPILER INDEX TSC BASIC

JAN 85 P43 ANNOUNCEMENT MICROWARE OS968K DOS

JAN 85 P43 KREINIK FLEX UTILITY M2 MODEM 1200 BAUD 6800 ASSEMBLY LISTING

JAN 85 P44 RUSSELL TEKTRONIKS DRIVER EMULATOR ARCADE 50 GRAPHICS BOARD FBASIC LISTING

JAN 85 P45 PIACENZA OUTPUT PL9 GEN STATEMENTS USING TSC MACROASSEMBLER TSC XBASIC LISTING

JAN 85 P47 PERIPHERAL TECHNOLOGY PRICE LIST

JAN 85 P47 HASSEN ARTICLE CONNECTING EPSON HX20 TO SWTPC SS50 BASIC AND 6800 ASSEMBLY LISTINGS

JAN 85 P49 LEONG FIX PATCH MOD TSC CAT UTILITY DISPLAY 3 FILES PER LINE 6800 ASSEMBLY LISTING

JAN 85 P49 STRUNK ARTICLE INTERFACE EPSON MX80 TO SWTPC CONNECTOR TABLE

FEB 85 P7 ANDERSON FLEX USER NOTES PROCKESS? 2MHZ NECESSARY? BARGAINS CRUNCH COBOL

FEB 85 P9 DIBBLE OS9 USER NOTES UTILITY MAKE C LISTING

FEB 85 P12 PASS C USER NOTES STRING HANDLING LIBRARY C FUNCTIONS C LISTINGS

FEB 85 P15 LUCIDO 68000 USER NOTES PORTABILITY SYNTAX LIBRARY PREPROCESSOR DATA TYPES

FEB 85 P17 BOLLINGER BOOK REVIEW SOFTWARE TOOLS IN PASCAL BY KERNIGAN AND PLAUGER TEXT FORMATTING LUNAR LANDER

FEB 85 P24 GROVES ARTICLE VIRTUAL DISK EXTENDED ADDRESSING OS9 LEVEL I 6809 ASSEMBLY LISTING

FEB 85 P26 6809 SINGLE BOARD COMPUTERS SARDIS ST2900 MICROKEY 4500

FEB 85 P29 ARMSTRONG FLEX UTILITY LOCAL RESIDENT COMMANDS EXTENDED ADDRESSING SWTPC DAT WHIMSICAL LISTING

FEB 85 P34 PASS ARTICLE COMPARE TSC AND MICROSOFT BASICS XPC

FEB 85 P42 JAMES LETTER LIGHT SWITCH PIA CONTROL USE 6809 ASSEMBLY LISTING

FEB 85 P43 FLEX EQUATES

FEB 85 P45 OADBY KEY IO FUNCTION KEYS LUCIDATA PASCAL AND 6809 ASSEMBLY ISTINGS

FEB 85 P48 JEFFREY CONSTRUCTION SS50 256K DYNAMIC RAM SS50 SCHEMATIC

FEB 85 P50 POUNDS UTILITY SSB DOS68.51 SINGLE DRIVE COPY 6800 ASSEMBLY LISTING

MAR 85 P8 ANDERSON FLEX USER NOTES COCO DEBATE USED EQUIPMENT GIMIX BIDEO DRIVER STARDOS BBS RX TX PL9 LISTINGS

MAR 85 P13 DIBBLE OS9 USER NOTES KEN KAPLAN INTERVIEW PROCESS PRIORITY C LISTING OS9 ON SWTPC

MAR 85 P16 PASS C USER NOTES STRING HANDLING FUNCTIONS C LISTINGS

MAR 85 P19 LUCIDO 68000 USER NOTES DATA TYPES DIRECT SPECIFIER 68020

MAR 85 P21 NAY REVIEW TMP FREEFORM FILER FILE MANAGER DBMS

MAR 85 P25 PASS FLEX UTILITY SWTPC MIRROR 6809 ASSEMBLY LISTING DISK TRACK FOR TRACK COPY

MAR 85 P29 WOLF ARTICLE BUILD YOUR OWN FAT MAC 512K RAM UPGRADE SCHEMATIC

MAR 85 30 GOULETTE FLEX UTILITY CAT 6800 ASSEMBLY LISTING

MAR 85 P40 AULICINO FLEX UTILITY ADDRESS MAIL LIST 6809 ASSEMBLY LISTING

MAR 85 P45 MCDANIEL ARTICLE 12 BIT A TO D AND D TO A CONVERTER AD567 TI SP1 BASIC AND PASCAL LISTINGS SCHEMATIC

MAR 85 P48 JONES FIX PATCH TAYLOR'S COPY.CMD FLEX FILE NUMBER ABORT 6800 ASSEMBLY LISTING

MAR 85 P51 JONES LETTER PATCH FIX PC PRINTER DRIVER LINE FEED 6800 ASSEMBLY LISTING

MAR 85 P51 CRAIG BASIC PROGRAMS FOR JPC AD16 A TO D SS30 LISTINGS

MAR 85 P54 FRASER FLEX UTILITY SETIME DATE 6809 ASSEMBLY LISTING

APR 85 P7 ANDERSON FLEX USER NOTES DISK COMPATABILITY DOUBLE DENSITY FLEX VERSIONS 68XX USERS GROUP SORTING

APR 85 P12 DIBBLE OS9 USER NOTES MEMORY ALLOCATION PROBLEM MEMORY MAP COCO C

APR 85 P14 PASS C USER NOTES C PROBLEMS MEMORY LIMITATIONS MICROWARE INTROL ETS C TUTORIAL INSTR.C UNTAB.C LISTINGS

APR 85 P19 LUCIDO 68000 USER NOTES MOTOROLA 68020 INSTRUCTIONS

APR 85 P21 MANN COCO USER NOTES COCO AND BUSINESS VIP WRITER

APR 85 P23 VOIGTS BASIC OS9 BASIC09 SORTING LISTING

APR 85 P27 REVIEW KBASIC LLOYD IO

APR 85 P28 WOLF MAC COLUMN ATARI TRACK BALL OAVONG HARD DISK

APR 85 P29 HOGLUND PATCH MOD DIRECTOR DISK DIRECTORY FLEX 6809 ASSEMBLY LISTING

APR 85 P30 HENRIQUES ARTICLE 512K MAC

APR 85 P31 HOFFMAN BIT SLICER OS9 UTILITIES PACK MAKE SDIR

APR 85 P42 GILCHRIST TAYLOR ARTICLE C.DRAGON SOFTWARE EXCHANGE SYSTEM BBS MODEM

APR 85 P45 MASSEN ARTICLE DISK INVENTORY BUILDCAT BASIC LISTING

APR 85 P46 LESTER PLEASANT PL9 CONVERT TELEWRITER BINARY FILES TO ASCII PL9 LISTING

APR 85 P48 1984 INDEX 68 MICRO JOURNAL

APR 85 P49 1984 INDEX COCO COLOR MICRO JOURNAL

APR 85 P51 FLEX UTILITY PRINTER PASS THRU FOR TSC XBASIC 6809 ASSEMBLY LISTINC

APR 85 P52 MORSE OS9 UTILITY DEBUGGING TOOLS LBSR SHOREGS SHOMEM 6809 ASSEMBLY LISTING

APR 85 P54 MORSE LETTER OS9 STDIN STDERR ARE NOT INDEPENDENT 6809 ASSEMBLY LISTING

APR 85 P55 ANNOUNCEMENT MOTOROLA DMA CONTROLLERS DYNAMIC RAM FLOATING POINT PROCESSOR 68440 68442 68450 4164 68881

MAY 85 P7 ANDERSON FLEX USER NOTES COMPARES SS50 TO OTHERS IBM MICROSOFT BASIC ATARI 800 FLIGHT SIMULATOR KBASIC

MAY 85 P11 DIBBLE OS9 USER NOTES COCO GAME WEASEL VIRUS BASIC09 LISTING

MAY 85 P12 PASS C USER NOTES C TUTORIAL VARIABLES CONSTANTS EXPRESSIONS FUNCTIONS STATEMENTS

MAY 85 P18 LUCIDO 68000 USER NOTES 68020 DESCRIPTION CONT. REGISTERS VIRTUAL OPERATION ADDRESSING TIMING

MAY 85 P20 ANDERSON REVIEW WHIMSICAL 6809 COMPILER SINE FUNCTION WHIMSICAL LISTING

MAY 85 P22 ELBERT ARTICLE ADA 68000 ADA LANGUAGE HISTORY

MAY 85 P26 VOIGTS BASIC OS9 NO PERMISSION FILE ATTRIBUTES SORT PROCEDURE BASIC09 LISTING

MAY 85 P29 MANN COCO USER NOTES UNINTERRUPTABLE POWER SUPPLY UPS STARDOS+ SPEL N FIX 2

MAY 85 P32 BRUMLEY USING FLEX STARDOS INTRODUCTION

MAY 85 P33 MAY ARTICLE APPLE MACINTOSH MAC

MAY 85 P41 GLENNON LETTER OBSOLESCENT SOFTWARE HARDWARE

MAY 85 P43 CASNEUF ARTICLE MOD FIX SWTPC 6809 OS9 EXTENDED ADDRESSING 2MHZ UNIFLEX TIMER SCHEMATICS

MAY 85 P47 LEE OS9 UTILITIES PRINT NAME 6809 ASSEMBLY LISTING

MAY 85 P51 STARK ARTICLE ROUNDOFF ERRORS IN COCO

MAY 85 P55 BAUER LETTER SS50 HIRES GRAPHICS

MAY 85 P55 LEMOINE LETTER PATCH FIX MIRROR.CMD BY PASS

MAY 85 P56 ADAMS LETTER PRINT.SYS 6809 ASSEMBLY LISTING PARALLEL PRINTER

MAY 85 P57 FIX PATCH DO.CMD BY SLAGHEKKE AND LOG .CMD BY YSSEL 6809 ASSEMBLY LISTINGS

MAY 85 P58 FORREST FLEX UTILITY HEX CONVERSION CALCULATOR 6809 ASSEMBLY LISTING

JUN 85 P7 ANDERSON FLEX USER NOTES MAILING DISKS OPERATING SYSTEMS STARDOS OUTCM2

JUN 85 P12 DIBBLE OS9 USER NOTES OS968K ERROR MESSAGES PROCESS PRIORITY MODULE HEADER INIT DATA MODULES

JUN 85 P16 PASS C USER NOTES TYPE CONVERSIONS VARIABLE INITIALIZATION POINTERS AND ADDRESSES POINTERS VS ARRAYS

JUN 85 P21 LUCIDO 68000 USER NOTES MAC RAM PRICES

JUN 85 P23 RAMBLINGS 6800 HISTORY

JUN 85 P27 ELBERT ADA 68000 LANGUAGE CHARACTERISTICS

JUN 85 P30 VOIGTS BASIC OS9 DISPLAY PDISPLAY BASIC09 LISTING PRINTER FORM FEED 6809 ASSEMBLY LISTING

JUN 85 P37 MANN COCO USER NOTES HOYT STERNS COLOR FORTH

JUN 85 P38 LYON ARTICLE DISK INCOMPATABILITY NEWDISK FLEX UTILITY 6809 ASSEMBLY LISTING

JUN 85 P40 DILDY ARTICLE FLEX HEX LOAD UTILITY PL9 LISTING S1 S9

JUN 85 P42 KITAZUME ARTICLE UPGRADE SWTPC 6809 F&D VIDEO EXTENDED ADDRESSING JCP SCHEMATIC 6809 ASSEMBLY LISTING

JUN 85 P45 SPRAY FILE COMPARE UTILITY MATCH WHIMSICAL LISTING

JUN 85 P47 LONE FIX PATCH MOD EDITBAS STRING FILE EDITOR BASIC LISTING

JUN 85 P48 HAM NET 68XX

JUN 85 P48 TALBOT LETTER MAC DEVELOPMENT SYSTEM FORTH

JUN 85 P51 MCDANIEL ADDENDUM A TO D D TO A FROM MAR 85 BASIC LISTINGS

JUL 85 P7 ANDERSON FLEX USER NOTES TUTORIAL SORTING DISK INCOMPATABILITY

JUL 85 P12 DIBBLE OS9 USER NOTES VIRTUAL MEMORY DAT LISP JOURNAL FOR IRS OR LOG

JUL 85 P16 PASS C USER NOTES STANDARD C LIBRARY IO MEMORY MANAGEMENT C LISTINGS

JUL 85 P20 LUCIDO 68000 USER NOTES MAC RAM UPGRADE HAZELWOOD DM256 OS968K STOP WATCH TIME.C LISTING

JUL 85 P25 ELBERT ADA 68000 MODIFIABLE EFFICIENT RELIABLE UNDRSTANDABLE PORTABLE REUSABLE

JUL 85 P28 VOIGTS BASIC OS9 FILE DUMP DISK IO BASIC09 LISTING KANSAS CITY BASIC COCO

JUL 85 P35 MANN COCO USER NOTES MODEM COMMUNICATIONS TUTORIAL

JUL 85 P37 GILCHRIST TAYLOR UNIX TOOLS DRAGON ENGINE INTROL C FLEX SHELL GREP LS C LISTINGS

JUL 85 P45 BIT BUCKET COCO TANDY RADIO SHACK TRENDS

JUL 85 P46 STANLEY LETTER INTERFACE KSR43 TTY TO SWTPC MPS SERIAL INTERFACE SCHEMATIC

JUL 85 P47 MUIR ARTICLE NRDY PROBLEM SS50C SYNCHRONIZATION PROBLEM

JUL 85 P48 RITCHIE LETTER BUG FIX PATCH LOG.CMD

JUL 85 P49 JONES LETTER PATCH FIX MOD TSC EDITOR CHANGE CHARACTER DEFAULTS

AUG 85 P5 ANDERSON FLEX USER NOTES COMPUTER FUN IBM PC JUST.C FLEX FILES .TXT .BIN FCB

AUG 85 P9 DIBBLE OS9 USER NOTES MODULE DIRECTORY OS9 USER GROUP PROBLEMS

AUG 85 P12 PASS C USER NOTES C VERSIONS PROGRAM CODE GENERATION DEBUGGING

AUG 85 P16 LUCIDO 68000 USER NOTES INSIDE MAC OS968K MAKE BUG SAMPLE.C LISTING

AUG 85 P21 ELBERT ADA 68000 PROGRAM UNITS PACKAGES

AUG 85 P25 VOIGTS BASIC OS9 FINDING FILES DISKLOOK OS9 UTILITY BASIC09 LISTING

AUG 85 P29 MANN COCO USER NOTES MODEM COMMUNICATIONS BBS RAINBORD

AUG 85 P30 BRUMLEY FLEX STARDOS COCO INSTALLATION

AUG 85 P36 PASS ARTICLE FLEX TYPE AHEAD INTERRUPT DRIVEN 6809 ASSEMBLY LISTING

AUG 85 P39 MARTIN ARTICLE COBOL NAME AND ADDRESS SYSTEM LISTING

AUG 85 P43 GROVES OS9 UTILITY SETIME MODULE 6809 ASSEMBLY LISTING

AUG 85 P47 GIMIX ANNOUNCEMENT 68020 CPU

AUG 85 P47 JONES LETTER FIX MOD PATCH SLAGHEKKE'S DO.CMD

AUG 85 P49 ADAMS LETTER FIX MOD PT69 ADD BATTERY BACKUP TO CLOCK SCHEMATIC

AUG 85 P50 OS9 SOFTWARE SOURCEBOOK CORRECTIONS

AUG 85 P50 GERWITZ LETTER PT69 HARD DISK SYSTEM

AUG 85 P52 RITCHIE LETTER BUG FIX LOG.CMD

SEP 85 P5 ANDERSON FLEX USER NOTES EDITOR FLEX FILES COPY TEXT FILES DUPLICAT FLEX UTILITY 6809 ASSEMBLY LISTING

SEP 85 P9 DIBBLE USER NOTES LEVEL II MODULE DIRECTORY VIRTUAL MEMORY USERS GROUP MICROWARE QA

SEP 85 P11 PASS C USER NOTES GUIDELINES PREPROCESSOR DECLARATIONS COMMENTS SYNTAX EXAMPLE PROGRAM C LISTING

SEP 85 P15 ELBERT ADA 68000 DATATYPES CONTROL STRUCTURES

SEP 85 P18 VOIGTS BASIC OS9 MEMORY ALLOCATION ERRORS RADIO SHACK C SECURITY

SEP 85 P20 LUCIDO 68000 USER NOTES MAC EXCEPTION VECTOR MANAGING RAM RESOURCES

SEP 85 P22 MANN COCO USER NOTES COMPUTER REVOLUTION

SEP 85 P23 LESTER PLEASANT PL9 KEY WORDS SHOPPING LIST PL9 LISTING

SEP 85 P25 SCHMITZ ARTICLE GUARANTEE 680X OPERATION WITHIN SPECS TIMING TABLES

SEP 85 P36 MARTIN COBOL NAME AND ADDRESS CONT. LISTING

SEP 85 P37 ARMSTRONG FLEX UTILITY SIZE PARSE WHIMSICAL LISTINGS

SEP 85 P40 SPRAY FLEX UTILITY CMD GENERATION WHIMSICAL LISTING

SEP 85 P43 TAYLOR GILCHRIST FILE DUMP C LISTING

SEP 85 P45 HALL INTERFACE SUMMAGRAPHICS BIT PAD TO SWTPC 6800 RS232 ACIA MPS

SEP 85 P46 JONES LETTER ENHANCE COPY CAT FLEX TO INCLUDE TIME CODE

SEP 85 P47 SILLANPAA LETTER FIX MOD UNIBOARD 6809 SBC 6809 ASSEMBLY LISTING

SEP 85 P48 OESTEN FIX PATCH FLEX RPTERR TO PRESERVE FILE NOT FOUND

SEP 85 P49 DREXLER ANOTHER SOLUTION FOR INCOMPATIBLE DISKS FLEX DISK DRIVERS 6809 ASSEMBLY LISTING PT69 SCHEMATIC

OCT 85 P5 ANDERSON FLEX USER NOTES EDITORS TANDY 1200 FLEX CHANGE DISKS WITH FILES OPEN ASCII VS HEX

OCT 85 P7 DIBBLE OS9 USER NOTES ATARI OS9? GIMIX 68020 HAZELWOOD UNIQUAD PROCESSES PROCS COMMAND C LISTING

OCT 85 P12 PASS C USER NOTES CODE GENERATION COMPARISON FUNCTION CALLS DEFINITIONS CODE OPTIMIZATION

OCT 85 P17 ELBERT ADA 68000 SUBPROGRAMS

OCT 85 P19 VOIGTS BASIC OS9 COCO OS9 PASCAL LEONARDO'S RABBITS PUZZLE XSCREEN

OCT 85 P22 LUCIDO 68000 USER NOTES TIME FIX MOD PATCH

OCT 85 P24 LESTER PLEASANT PL9 EPSON FX80 LEFT MARGIN PL9 LISTING

OCT 85 P25 MANN COCO USER NOTES CASSETTE TAPE LABEL MAKER BASIC LISTING

OCT 85 P26 GROVES OS9 SETIME MODULE FIX MOD PATCH 6809 ASSEMBLY LISTING

OCT 85 P29 MARTIN COBOL NAME AND ADDRESS SYSTEM CONT. LISTING

OCT 85 P38 TAYLOR ARTICLE LFEX DISK SECTOR INTERLEAVE

OCT 85 P41 RAMBLING AROUND NCC CIMIX 68020

OCT 85 P44 ANNOUNCEMENT MICROWARE OS9 SUPPORT FOR HITACHI HD63484 GRAPHICS CONTROLLER

OCT 85 P46 GIMIX ANNOUNCEMENT 32 BIT MC68020 SINGLE BOARD COMPUTER

OCT 85 P47 JONES LETTER PATCH FIX FLEX DOCMND BUG

OCT 85 P48 THOMAS LETTER FLEX PRINTER SPOOLER BUG SWTPC

OCT 85 P51 VOICTS REVIEW DYNACALC SPREADSHEET FOR COCO

OCT 85 P52 VOIGTS REVIEW COCO OS9 PASCAL

NOV 85 P7 ANDERSON FLEX USER NOTES PAT TEXT EDITOR DESCRIPTION

NOV 85 P16 DIBBLE OS9 USER NOTES LEVEL II PROCESSES BENCHMARKS HOT SCOOPS MCI MAIL

NOV 85 P19 PASS C USER NOTES MCCOSH C COMPILERS DESCRIPTION

NOV 85 P24 ELBERT ADA 68000 PACKAGES

NOV 85 P27 VOIGTS BASIC OS9 SHELL ALPHABETIZE DIRECTORIES C LISTING

NOV 85 P29 LEWIS UNIFLEX USER NOTES

NOV 85 P35 GONDON ISAM INDEXED SEQUENTIAL ACCESS METHOD BASIC AND 6809 ASSEMBLY LISTINGS

NOV 85 P43 WILLIAMS RAMBLINGS AND SUCH JUST SS50 68XX SERVICE DEPARTMENT 512K COCO MAUSTANG 68020 68008 CPU

NOV 85 P45 VOIGTS REVIEW BOOK COMPLETE RAINBOW GUIDE TO OS9 BY PUCKETT AND DIBBLE

NOV 85 P45 VOIGTS REVIEW BOOK OS9 USER NOTES BY DIBBLE

NOV 85 P48 WILLIAMS DATACOMP MUSTANG 020 SBC

NOV 85 P48 SARDIS TECH LETTER ST2900 6809 SYSTEM

NOV 85 P50 JONES LETTER INNER WORKINGS OF TSC XBASIC LINKED LIST TOKENS

NOV 85 P52 DILDY PROGRAM TO SOLVE TOWERS OF HANOI PL9 LISTING DEMONSTRATE RECURSION

NOV 85 P52 TROLLOPE FIX MOD PT69 SBC TO GIVE MORE RAM 68764 SCHEMATIC

DEC 85 P7 ANDERSON FLEX USER NOTES BASIC PASCAL C FORTH PL9 MUSICAL ASSEMBLER COMPARISONS

DEC 85 P10 DIBBLE OS9 USER NOTES OS968K

DEC 85 P11 LUCIDO 68000 USER NOTES 68000 ASSEMBLY LANGUAGE

DEC 85 P13 PASS C USER NOTES INTROL C COMPILER MCCOSH C B TREE SEARCH ALGORITHM DETAB C LISTING

DEC 85 P17 LEWIS UNIFLEX USER NOTES FLEX FOR UNIFLEX TSCC COMPILER

DEC 85 P18 ELBERT ADA 68000 CONCURRENT PROCESSING TASKS

DEC 85 P21 VOIGTS BASIC OS9 BACKUP DISK NAME CHANGE C LISTING

DEC 85 P22 HOFFMAN ARTICLE USING K BASIC LLOYD IO BASIC COMPILER

DEC 85 P25 BROOKER REVIEW MICROBOX II 6809 SBC

DEC 85 P26 CONDON ISAM CONT. 6809 ASSEMBLY LISTING

DEC 85 P35 GERWITZ FLEX UTILITY UPDATE FUNCTION BY GOADBY FEB 85 6809 ASSEMBLY LISTINGS

DEC 85 P37 ROBINSON ARTICLE EXPAND MVME201 TO 1 MEGABYTE RAM SCHEMATIC

DEC 85 P39 BRUMLEY USING FLEX STARDOS UTILITY COMMAND SET PRINTER DRIVER CUSTOMIZATION COCO

DEC 85 P41 MANN COCO USER NOTES DISK SYSTEMS

DEC 85 P42 ANDERSON REVIEW CEDRIC SCREEN ORIENTED TEXT EDITOR FOR FLEX

DEC 85 P43 DESCRIPTION SOLVE OS9 DEBUGGER ASSEMBLER DISASSEMBLER

DEC 85 P47 STARKITS DIFFERENCES BETWEEN FLEX AND STARDOS

DEC 85 P48 LARRIMORE FLEX PATCH UTILITY HELP 6809 ASSEMBLY LISTING

WESTCHESTER Applied Business Systems, Inc
2 Pea Pond Lane, Briarcliff Manor, New York 10510

January 20, 1986

Don Williams
'68 Micro Journal
5900 Cassandra Smith
Hixson, Tenn 37343

Dear Don,

We are in the final stages of testing XDMS-IV, the newest version of the XDMS Data Management System. One of the enhancements in this version which was requested by many users, is the use of random files for paging. Paging, in XDMS, is used to swap out memory to disk to allow manipulation of large files which do not fit into memory.

One of my objectives with this version was to ensure compatibility with both FLEX and SK*DOS (formerly STAR DOS). I had spoken to Peter Stark of STAR KITS some time ago, at which time we discussed the notion of loading the index sector into RAM to reduce head movement. Happily, Peter has implemented this in the form of a RANDOM command. I have run some benchmarks which proved quite interesting.

The benchmark is 1000 records each with an effective length of 64 characters, ogi a 64K file. Internally records are compressed to 54 characters. Keys were arranged so that the primary sort key repeats within the secondary key, thus rendering a pseudo-random sequence. Times were recorded to load, and to load and sort the file using several different mechanisms. A 1MHz SWTPc 6809 with 8" DMAF1 DSSD's was used.

Ogi I Method	Load file only --	Load and Sort ---
FLEX with random paging	0:05:20 (320sec)	3:24:00 (12240sec)
SK*DOS w. random paging	0:03:56 (236sec)	2:09:00 (7740sec)
Direct Sector paging	0:01:42 (102sec)	0:45:30 (2730sec)
RAM load/sort keys only	0:00:47 (47sec)	0:02:21 (141sec)

Indications are that SK*DOS is 1.53 times faster than FLEX on random file handling. Direct sector paging (as offered with XDMS today) is about 4.5 times faster. The RAM load and sort numbers are included as a comparision with other system benchmarks. Typically these are run under ideal conditions with small records, and in RAM. In all cases actual sort times are derived by subtracting the load times.

XDMS-IV will be offered with both random and direct paging capability. Users may select random for one of two reasons; 1) Only one drive is available (direct paging uses a full drive) or 2) paging is to be done on a hard disk. While numbers are not available, hard disks typically run 5-10 times faster than floppies. We endorse SK*DOS for random paging, and for general use. It is a nicely done, user friendly DOS. Announcement of XDMS-IV will follow shortly.

Best Regards,
Bill Adams

Dear Sir;

I would appreciate more articles on Sculptor. I am specifically interested in weather it is command driven, or if it uses menus, and fill in the blank screens.

I work for LTV Aerospace in Dallas, and we are converting to 4th generation languages now. I am interested in learning if Sculptor is powerful enough, and easy enough to use to be considered by our company.

We are now looking at PC Ramis and Rbase 5000. Thanks for your articles.

Sincerely,

Cliff Rushing
1010 E. Arkansas Ln. Apt. 145
Arlington, TX 76014

Dear Don:

First let me say that I bought my first copies of your magazine a few weeks ago from Acorn Computer Systems in Milwaukee and am so pleased to find your magazine that I am enclosing my check for a 2 year subscription.

My current system consists of a 64K COCO II, a Taxan monitor, WORD-PAK II for 80 column display a 40 track system drive and two 80 track double sided drives, and a Gemini 10X printer. I use this system along with UDRI's business software running under COCO FLEX to run a mail order business.

This brings me to the reason for this letter. URDI has discontinued support for their Color Computer software due to low sales. They will continue to sell their Color Computer software with the source code included, but will not fix any program errors or do any updating of the programs.

I have found this most disappointing as there are several bugs in the various programs. Some of these I have been able to correct and would like to pass along the fixes to others who may be having the same problems.

The first fix is to the program AR04.SRC which prints the invoices. When using this program in conjunction with the inventory data file STOCK.DAT it will not always calculate the correct cost of goods sold. To fix this problem, line 4155 of the source code which reads as follows: CG=0 : FOR PL%=1 to 6 : CG=CG+CG(PL%) : NEXT PL% : FU(27,3)=CC. This should be changed to read CG=0 : FOR PL%= 1 to LI% : CC=CG+CG(PL%) : NEXT PL% : FU(27,3)=CG. Changing the 6 to LI% will correctly calculate the cost of goods sold for every invoice.

The following program lines can be added to the invoice printing program AR04.SRC to update the date last sold and YTD sales in the customer data file which are not currently updated.

```
5012 IF TY$ = "1" THEN FU$(14,1) =JD$
5013 IF TY$ = "1" THEN FU(16,1)=FU(16,1) + ((SC+ST+IA)*TY)
5014 IF TY$="2" THEN FU(16,1) = FU(16,1) + ((SC+ST+IA)+TY)
5018 IF FU(16,1)<0.01 THEN FU(16,1) = 0
```

If the above lines are added then line 5015 in the program AR03.SRC must be changed to read FU(15,1) = FU(15,1)-CH. If this line is not changed the YTD sales will be updated twice.

In the program AR07D.SRC the following must be changed in order to let the Accounts Receivable posting file records be added to the monthly archive file. In lines 1090 and 1140 the statement FU%(7,1) = X2%(2) must be changed to read FU%(7,1)=0 and in lines 2110, 2325, and 2345 the statement FU%(7,1)=X2%(3) must be changed to read FU%(7,1)=0.

In the accounts payable program AP05D1.SRC the following lines must be added in order for the program to add account numbers to the posting file records:

```
201 OPEN OLD "ACCPAY.DAT" AS 1
202 FIELD #1,78 AS F1$,120 AS F2$,50 AS F3$,4 AS F4$
203 GET #1, RECORD 1
204 Q$(1)=LEFT$(F3$,10) : Q$(2)=MID$(F3$,11,10) :
Q$(3)=MID$(F3$,21,10)
205 Q$(4)=MID$(F3$,31,10) : Q$(5)=MID$(F3$,41,10)
206 FOR QZ=1 TO 5 : Q$(QZ)=LEFT$(Q$(QZ)+Z$,10) : NEXT QZ
207 CLOSE 1
```

There are some other problems with some of the programs but I have not yet had enough time to get them fixed. I would be willing to correspond with anyone else who has bought any of the Color Computer programs from UDRI and has had problems and fixed them or is currently having problems but does not know how to solve the problem.

If anyone has the compiled programs but not the source codes I would advise you to purchase the source codes for all of your compiled programs in order to be able to fix them to work properly.

Correapondence can be mailed to the following address:

Dale Przybyl
2438 North 48th Street
Milwaukee, WI 53210

I can be reached by phone in the evening or weekends at (414)-873-7162

Sincerely,

Dale Przybyl

** Editor's Note: We, 68 Micro Journal, would greatly appreciate if each of you would also send us the fixes so that I can include them in our files as well.

Also it can be assumed that these same errors are present in some others version (for other machines). So if you can get the fixes to us, we can attempt to insure that the information is published, in order that ALL can be made aware.

Thanks.

DHW

- - -

COMPUSENSE LTD.
Computer Systems Consultants

P.O. Box 169
2860 Green Lanes
London N13 5TN
Tel: 01-882 0681
01-882 6936
Telex: 8813271 G

THE THIRD 6809 DRACON/COLOR SHOW

The weekend of November 22nd and 23rd 1985 was a special day for all British 6809 owners, and indeed for many of our European friends. Once again it was time for the 6809 show at the Royal Horticultural Halls.

While this show is not the largest computer show in the U.K. by any means, it is the ONLY British show which is devoted soley to owners of 68xx based computers.

This show happens once every six months at the same venue, so if any 6809 owner within travelling distance has not been to the show yet then shame on you!

There were about seven thousand vistors over the two days of the show in spite of typically hostile English weather.

Considering that the DRAGON and TANDY COLOR computers have not had an easy ride in the U.K. and Europe the number of visitors was a pleasant shock for the new exhibitors.

So what was going on at the show?

Well, there were about forty stands selling computers, hardware and software. The offerings were varied, from cassette games for the TANDY to sophisticated computers and processor cards.

There were piles of cassette software for the DRAGON and TANDY COLOR of every shape and description. I pretend not to like games, but somehow I always end up at the Microdeal stand who have over 100 titles for the DRACON and TANDY computers. Microdeal have a travelling video arcade with computers, screens and joysticks where the young, and not so young, afficionados gather to try the latest releases. Microdeal is well organised and they brought and sold thousands of tapes. A bit noisy though and welcome relief to pass around to some of the more serious folk at the show.

In fact, this 6809 show had the best "serious" representation of any of the preceding shows. I remember Compusense being the only serious company at the first show. (You may find this hard to believe but until Compusense stuffed FLEX in peoples faces the majority of DRAGON owners hadn't even heard of it!) This show had many of my favourite names and faces appearing :-

Cambridge Micro Systems were showing their range of industrial Eurocards including a 6809 based second processor card for the BBC computer. The BBC is based on a 6502, poor thing, and Phil's card gives the BBC fraternity the ability to use their computer with some decent FLEX software.

Just a few yards away from Phils stand was the Microconcepts stand. Jim and Carole Rew have a fine company with excellent hardware and software support.

Dr. David Rundle the designer of Microbox 2, recently reviewed in the pages of this journal was also in attendance. Just as visible was Adrian of Silicon Fen software who has produced a set of cross "c" compilers for FLEX - when do I get a review copy Adrian? I think that there will be more news from Microconcepts soon on a new computer.

Another name which is well known to all 68xx owners is that of Nigel Bennie of Lucidata. Surely everyone has heard of LUCIDATA PASCAL. Nigel with wife Eileen and son were proudly showing off the "DRAGONS CLAW". The Claw is designed to allow DRAGON owners access to all the rather nice peripherals which BBC owners have in this country. Nigel was going well all weekend demonstrating a robot arm and camera interface to interested spectators.

Further up from Lucidata was another interesting stand. Jim Anderson of Andtek Design was demonstrating the "PLUS" interface for DRAGON and also an expansion interface kit for the DRAGON at a remarkably low price. The PLUS interface combines static ram, two RS232's, a disk controller and a battery backed clock to give real power to the DRAGON.

Further around the hall there were the club stands. The 68' Micro Croup were very much in evidence and Ted Bacciarelli and Roger Parrish who were manning the stand on Sunday. By the way the pile of 68 journals on the stand were items of hot interest! Close by were the National Dragon Users Group.

Finally of course there was our own stand, COMPUSENSE. We were showing FLEX, the DRAGON and demonstrating software and hardware. On the software side Stan Opyrchal was showing off our new word processor for FLEX and also modems and communications software for the DRAGON and FLEX. On the hardware front there were Eprom programmers, serial interfaces, expansion boxes and DRAGONPLUS. The DRAGONPLUS hardware upgrades a DRACON to 128k RAM with an onboard 6845 display using its own 2k static RAM. If that wasn't enough to grab peoples interest we were also demonstrating a 10 meg hard disk attached to a DRAGON (running FLEX of course).

Every exhibitor who had something worth showing was swamped!

What of the future? I am sure that every one of the companies above will be back in March 1986 will be back for the next show. I look forward to the time, maybe next time, when the REAL 68xx companies will outnumber the other guys. There are at least ten more companies I would still like to see at the next show, and I know that they all read this magazine!

DRACON owners represent about 30,000 active users of 6809 based computers, going by ABC rating of "Dragon User" magazine. Of course the majority of owners are just "playing games" but a significant number bought the computer precisely because it contains a 6809 chip. What I have found is that many Dragon owners turn out to be design engineers or somehow involved in the real microcomputer busainess. So while young Johnny is off playing games at the Microdeal stand his father is quietly evaluating hardware and software at our stand. Sometimes it happens the other way around!

Yes, we made many new friends and renewed old aquaintances at the show. Yes, it was certainly worth attending for us and for the public alike. I enjoy meeting my customers and letting them judge both me and my products face to face. Incidentally the show also gives us a welcome boost in sales leads and is always finacially rewarding. Need I say more?

T Opyrchal

DATA-COMP

NEWS RELEASE

HEAVY DUTY
SWITCHING POWER SUPPLY

For a limited time Data-Comp will offer a HEAVY DUTY switching power supply, for those wanting to do your own thing with hard disk systems, etc. The specifications are shown below. Note that this is a price far below normal prices for supplies of this quality.

Make: Boschert -- Size: 10.5x5x2.5 Inches-including heavy mounting bracket/heatsink

Rating: In: 110/220 ac (strap change)
Out: 130 watts

Outputs: +5 volts - 10.0 amps
+12 volts - 4.0 amps
+12 volts - 2.0 amps
-12 volts - 0.5 amps

Mating Connector: Terminal strip
Load reaction: Automatic short circuit recovery

PRICE: $59.95 + $7.50 S/H
2 or more $49.95 ea. + $7.50 S/H each

Also: Dyaan made Diskettes - 8" SSDD Box of 10 - $18.00
Data-Comp will pay Shipping on Diskettes in the U.S.A. & Canada.

Data-Comp
5900 Cassandra Smith Rd.
Hixson, Tn. 37343
(615) 842-4600

The Mustang - 020 Saga
or
A Day with a Mustang - 020

Many of you have called wondering what the Mustang-020 complete system looks like. Well we decided to make a few pictures of our Mustang-020 system, in actual use in our office. But, we ran into some problems. It was difficult getting it free, for a picture session. So we had to make do with the best we could. Hope you can figure out which is the Mustang-020.

First, as you can see, I must have a little leverage, to keep others (see other pictures) from stealing it away, while I am using it. We have about 10 other systems in our main office, but everyone wanted to ride the Mustang (a little joke). Seriously, we solved that problem - we put 19 more terminals on the Mustang-020. Now everyone rides it at the same time. Sorts like a Merry-Go-Round, but lots more fun! Except for one little thing. It is so darned fast that everyone complains because his tasks finishes too soon. Oh, well.

Mary does all the heavy stuff, from the front office on the Mustang-020. We would have made this picture from her office, and her CRT, but she, like all the rest wanted her picture taken with the main system (can you find it?)

Chris (S.E. MEDIA chief) also wanted to be photographed with the Mustang-020 system showing. Seems everyone around here wants to be a name dropper. Anyway Chris is entering an order for another Mustang system, into - you guessed it, the Mustang-020. Looks practically like the darn things smiling, don't it?

Allyson, Mary's daughter and my grand-daughter is also a great fan of the Mustang-020. As you can see, using the Mustang-020 is 'childs play' (not another little joke, this thing is just that much fun!)

Some asked about the ease of service (we have not had to do any yet.) Anyway, as you can see, Tom, our service manager finds it s 'piece of cake'! Guess who got to eat the cake later?

And for you skeptics, as you can see, even Charlie, our vice-president in charge of security, finds it dog-gone easy to operate. However, please accept our apology if you get a listing of dog bone burial plots, rather than your subscription renewal notice, in the next few months. Charlie, keeps merging the wrong files (no fault of the Mustang-020!) I think it is because he keeps renaming the 'Kernel' - 'Kennel'. Oh, well again.

So there you have it. A very complete day in the life of a Mustang-020 from Data-Comp. With the large variety of 'HHLs' available for the Mustang-020 (see advertisment) (please), developing new or porting old applications is made much easier.

Better get your order in before the price goes up. Which it will probably do about the second quarter of this year. And of course, we don't want to take any of the fun out of it, so I'm telling you now.

DMW

MVME101 TERMINAL CONFIGURATION

by Ray Robinson
Speech and Language Research Centre
School of English and Linguistics
Macquarie University
North Ryde 2113
NSW Australia

INTRODUCTION

The MVME101 is a VME bus 68000 single board computer card from MOTOROLA. It can be used as a stand alone processor or controller, or in conjunction with a disc controller and some memory, as a disc based computer. MOTOROLA can supply a system called the MVME315 kit which contains a MVME101 CPU card, a MVME201 256K DRAM card, a MVME315 disc controller, a boot ROM, and the VERSADOS disc operating system. The CPU card has 2 aerial ports and 1 parallel printer port.

The WYSE50 terminal has a smoothe scroll function which did not work with the MVME315 kit as supplied, so this is what is required for handshaking on the serial port. The terminal needs to control the speed at which the CPU is sending data, during the smoothe scroll, as the CPU can send much faster than the smoothe scroll speed, even the smoothe X8 mode (there are 4 smoothe scroll speeds and 1 jump scroll mode). The WYSE50 can control the CPU sending speed in 2 ways. It can stop the CPU by making the RS232 signal DTR go FALSE, and/or by sending control S (^S) the ASCII halt character (also called XOFF), on the serial data line. The CPU can then be told to start transmission by making DTR TRUE, and/or by sending control Q (^Q) the ASCII start character (also called XON). This is called XON-XOFF protocol.

XON-XOFF PROTOCOL

VERSADOS comes configured with control W (^W) as the start and stop character and a BREAK (^C) to abort the transmission. Use the SYSGEN utility to change this to XON-XOFF (^S^Q) and leave BREAK the unchanged. Follow the SYSGEN example in the manuals, and copy all the relevent files to user area 9100 on the disc. (My COPY utility has a non operative C option, so I had to rename some of the files to VME101). Then edit the file 9100.VME101.SYSCMD.CD to change the variables: TCP$XOF to hex 13 (^S the XOFF character), TCP$XON to hex 0 (this allows any character to start transmission including ^Q), and leave TCP$BRK as hex 3 (^C the ASCII BREAK character)(you can change it if you want to). Change the logon message REVNUMBR to 4.3:a to indicate the change. Copy the new VERSAOOS.SY to area 0000 and reboot.

The smoothe scroll now works nicely at all 4 speeds and the listings can be stopped with ^S and started with ^Q (or any key) and aborted with BREAK (^C).

I also made another version (4.3:b) which uses ESCAPE-ESCAPE as my FLEX system does, to see if I liked the old familiar control sequences.

DTR CONTROL

The RS232 cable should be wired as in Diagram 3. When the WYSE50 uses the DTR signal only, then VERSADOS responds with a DEVICE NOT READY error!

The same file we edited before, gives a clue to the problem. The parameter TCP$CTRL allows the use of the RS232 CTS signal to control the transmission. Figure 4.9 of the MVME101 M68000 MONOBOARD MICRO COMPUTER USER'S MANUAL sheet 8 shows that the MC68661 EPCI (Enhanced Programmable Communications Interface) (ACIA or UART to me) chip U52 has 2 patch areas for the RS232 signals but will force CTS and DTR to be connected together. Change the card such that the control signal from the terminal only goes to CTS on the EPCI.

The changes involve cutting 2 tracks, adding 2 pieces of wire, and repatching. First locate the track which goes from U58 pin 11 to U52 pin 22, and cut it near C35 next to U52 pin 22. Then under the socket of U52, the track continues from U52 pin 22 to patch area K9 pin 6. Cut this track under U52. Using a piece of thin insulated wire (wire wrap wire is ideal) join the track before the first cut to after the second cut. Check with an ohmn meter that U58 pin 11 goes to K9 pin 6, and does NOT go to U52 pin 22. See Diagram 1. Now connect U52 pin 22 to the earth end of C35. Check with the ohmn meter. So now DTR on the EPCI is earthed, giving permanently TRUE status, and the terminal control line goes to patch area K9.

Diagrsm 2 shows the patching on K9 to connect CTS on the EPCI to the buffer U58, and the patching on K7 to connect the RS232 plug P3 pin 20 (DTR) to the buffer U58 pin 13 (CTS). Diagram 3 shows the RS232 cable connections between the WYSE50 and the MVME101. Go into the setup menu and set the WYSE50 to use DTR control only. When a listing now occurs, the terminal's DTR control line is connected to the EPCI's CTS input and the smoothe scroll works fine.

REPLECTIONS

I don't use the smoothe acroll much but it is handy for scanning listings. However, the exercise is useful in becoming fsmiliar with the SYSGEN utility and examining the features of the VERSADOS disc operating system. The MVME101 processor board now haa the facility to control the CPU sending speed by XON-XOFF protocol (or any other character protocol). Not only doea the syatem allow smoothe scroll, but this will allow serial printers uaing XON-XOFF to be connected, and communicationa programs which use charcater protocol to function. The system also allows control of the CPU sending speed electricaly by the RS232 signal DTR (or CTS). The smoothe scroll can also be controlled by the DTR line and a serial printer using electrical control can now be connected to the other port (if slmilar changes are made). Since the DTR control now works, I can use the FLEX ESCAPE/ESACPE control of listings if I wish.

END

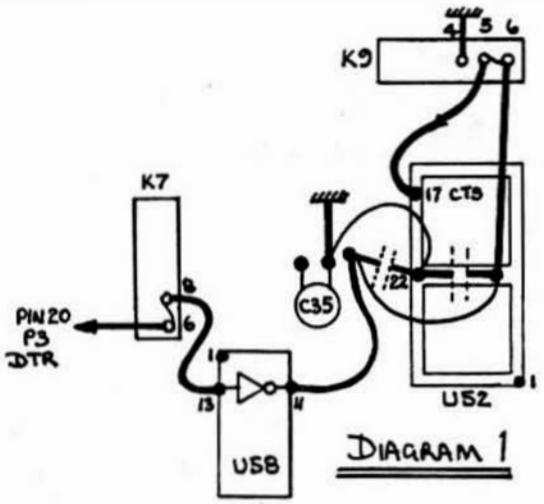


DIAGRAM 1

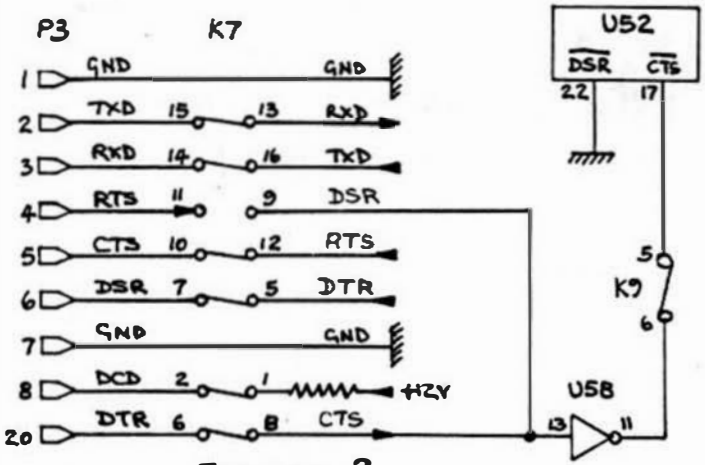


DIAGRAM 2

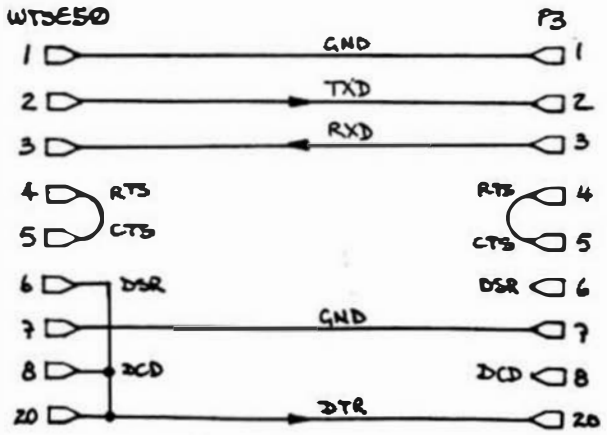


DIAGRAM 3

Micro 68 Journal
5900 Cassandra Smith
Box 849, Hixson, Tenn.
USA 37343

Gentlemen:

Enclosed are a couple of utility programs written in PL9 for you to publish if you like.

One is called DCAT; it prints a directory of drive 1, three up, sorted by date, with the most recent file last. How often have you wondered what you called that temporary file you created yesterday, or how recently a backup disk was updated ?

The source is written for the international Metric date standard (YY-MM-DD), but simple instructions to change this are included. Note that only 2 digits are provided for file size; the output will look a little funny for files larger than 99 sectors. There's no way to fit a listing with 3 digits in an 80 column screen.

The other program, LA (for List-All), lists ALL files on drive 1, one after another, honouring the FLEX pause feature.

Note that "all" includes .BIN and .CMD files, and that non-printable characters aren't translated into something innocuous (like "."'s), because my memory-mapped video board is able to display them. This shouldn't be a problem, because this program is normally used with disks of text or source files. It might drive some terminals around the bend, however, so you may wish to modify the source to change either or both of these features.

I use this program to divide the contents of a too-full disk into two disks, according to category. I first do a "P,CAT" of the disk, then do an "LA" of it with the printout in hand, marking the filenames to be off-loaded. Finally, I use the excellent Taylor-Puglia copy program (see P17, 68MJ, Oct/84), with the "Prompt before copy" and "Zap source" options, to effect the split.

Both of these programs use the library routines "FLEXDIR.LIB" (included), because the Windrush "FLEX.LIB" routines don't access the directory functions. Neither program is written to run in the UCS ($C100), although LA could probably be modified to do so. DCAT, however, requires too much memory for its data storage. At present, both programs store their data at $6000 so they can be run co-resident with PL9 and its debugger.

I hope you and your readers find these programs useful.

John Scott
Box 448, Lakefield
Ontario, CANADA
KOL 2HO

- - -

```
/* LA.PL9  84 01 18 */
/* LISTS ALL FILES ON DRIVE 1 */

AT $6000: BYTE DIRFCB(320);
AT $6200: BYTE LISTFCB(320);

INCLUDE FLEX.LIB;
INCLUDE FLEXDIR.LIB;

PROCEDURE CHROUT (BYTE CHR);
ACCA=CHR; CALL $CD18; ENDPROC;

PROCEDURE CRLF (BYTE N);
WHILE N > 0 BEGIN
  CALL $CD24;
  N=N-1; END;
ENDPROC;

PROCEDURE MAIN: BYTE I, TMP;
CRLF(1); DIRFCB(3)=1; OPEN_DIR(.DIRFCB);
REPEAT;
  READ_DIR (.DIRFCB);
  IF DIRFCB(1)<>0 THEN BREAK; /* END OF DIR ? */
  IF DIRFCB(4) > 0 THEN BEGIN  /* DELETED CHK */
    I=3; REPEAT;
      TMP=DIRFCB(I); LISTFCB(I)=TMP; /* COPY NAME */
      IF I <> 3 THEN CHROUT (TMP); /* PRINT IT TOO */
      IF I = 11 THEN CHROUT '.';
      I=I+1;
    UNTIL I=15;
    CRLF(2); OPEN_FOR_READ (.LISTFCB);
    REPEAT; /* PRINT THE FILE */
      TMP=READ(.LISTFCB);
      IF LISTFCB(1)<>0 THEN BREAK; /* EOF ? */
      IF TMP=$0D THEN CRLF(1); /* FOR FLEX PAUSE */
        ELSE CHROUT(TMP);
    FOREVER;
    CRLF(2); CLOSE_FILE (.LISTFCB);
  END;
FOREVER;

/* FLEXDIR.LIB    DIRECTORY FUNCTIONS  85 01 22 */
```

```
ASMPROC OPEN_DIR(INTEGER);
/* .FCB */
   GEN $AE,$62;          /*          LDX   2,S       */
   GEN $86,$06;          /*          LDA   #6        */
   GEN $A7,$84;          /*          STA   ,X        */
   GEN $7E,$D4,$06;      /*          JMP   FMS       */


ASMPROC READ_DIR(INTEGER);
/* .FCB */
   GEN $AE,$62;          /*          LDX   2,S       */
   GEN $86,$07;          /*          LDA   #7        */
   GEN $A7,$84;          /*          STA   ,X        */
   GEN $7E,$D4,$06;      /*          JMP   FMS       */

/* GCAT  B4 09 26  DIRECTORY SORTED BY DATE */

AT $6000: BYTE DIRFCB(320);
AT $6200: BYTE LISTFCB(320);
AT $6400: BYTE SWAPFLG(1);
AT $6403: BYTE DATA( 5000);

INCLUDE FLEXDIR.LIB;


PROCEDURE CHROUT (BYTE CHR);
ACCA=CHR; CALL $CD18; ENDPROC;


PROCEDURE CRLF (BYTE N);
WHILE N > 0 BEGIN
  CALL $CD24;
  N=N-1; END;
ENDPROC;


PROCEDURE SWAP (INTEGER J);
BYTE TMP, I;
I=0; REPEAT;
  TMP= DATA( J);
  DATA( J)= DATA( J+ 24);
  DATA( J+ 24)= TMP;
  J=J+1; I=I+1;
UNTIL I= 24;
SWAPFLG= 1;
ENDPROC;


PROCEDURE SPC (BYTE N);
WHILE N>0 BEGIN
  CHROUT( ' );
  N=N-1; END;
ENDPROC;

/* OUTPUT A BYTE AS 2 DIGITS,
NO LEADING ZERO SUPRESSION */

PROCEDURE DECOUT (BYTE VAL);
BYTE CHR;
CHR= '0;
WHILE VAL> 9 BEGIN
  CHR=CHR+1;
  VAL= VAL- 10; END;
CHROUT( CHR);
CHROUT( VAL+ '0);
ENDPROC;


/* OUTPUT A BYTE AS 2 DIGITS,
WITH LEADING ZERO SUPRESSION */

PROCEDURE DECOUT2 (BYTE VAL);
BYTE CHR;
CHR= '0;
WHILE VAL> 9 BEGIN
  CHR=CHR+1;
  VAL= VAL- 10; END;
IF CHR= '0 THEN CHR= ' ;
CHROUT( CHR);
CHROUT( VAL+ '0);
ENDPROC;


PROCEDURE MAIN;
INTEGER J;
BYTE MAXFIL, K, I, L, TMP;
CRLF(1); DIRFCB(3)=1; OPEN_DIR(.DIRFCB);
MAXFIL= 0;
REPEAT;              /* GET THE DATA TO ARRAY */
  READ_DIR (.DIRFCB);
  IF DIRFCB(1)<>0 THEN BREAK; /* ERROR CHK */
  IF DIRFCB(4) > 0 THEN BEGIN /* DELETED CHK */
    I= 0;
    REPEAT;
      DATA( MAXFIL* 24+( I))= DIRFCB(I+ 4);
      I=I+1;
    UNTIL I= 24;
    MAXFIL=MAXFIL+1;
  END;
FOREVER;
REPEAT;              /* SORT BY DATE */
  SWAPFLG= 0;
  J=0; K=0;
  REPEAT;
    IF DATA( J+ 23) = DATA( J+ 47) THEN BEGIN
      IF DATA( J+ 21) = DATA( J+ 45) THEN BEGIN
        IF DATA( J+ 22) > DATA( J+ 46) THEN SWAP( J);
      END;
      ELSE IF DATA( J+ 21) > DATA( J+ 45) THEN SWAP( J);
    END;
    ELSE IF DATA( J+ 23) > DATA( J+ 47) THEN SWAP( J);
    J= J+ 24; K=K+1;
  UNTIL K= MAXFIL- 1;
UNTIL SWAPFLG= 0;
J=0; K=0; L=0;       /* PRINT IT OUT */
REPEAT;
  I=0;
  REPEAT;            /* THE FILE NAME */
    TMP= DATA( J + I);
    IF TMP= 0 THEN TMP= ' ;
    CHROUT (TMP);
    IF I = 7 THEN CHROUT ('.);
```

```
    I=I+1;
  UNTIL I= 11;
  CHROUT( ' );
  DECOUT2( DATA( J+ 18)); /* SIZE */
  CHROUT( ' );
  /* CHANGE THE ORDER OF NEXT STMNTS
     TO GET MM-DD-YY */
  DECOUT( DATA( J+ 23)); /* YY */
  CHROUT( '-');
  DECOUT( DATA( J+ 21)); /* MM */
  CHROUT( '-');
  DECOUT( DATA( J+ 22)); /* DD */
  L=L+1;
  IF L= 3 THEN BEGIN
    L=0;
    CRLF(1); END;
  ELSE SPC( 2);
  J= J+ 24;
  K=K+1;
UNTIL K= MAXFIL;
```

109 ARD ST.
WELLINGTON
NEW ZEALAND.
(Tel WGTN NZ 850881)

22nd November 1985

Don Williams Sr.

Computer Publishing Center
68 Micro Journal
5900 Cassandra Smith Rd.
Hixson, Tn. 37343
U.S.A.

Dear Don,

Please find enclosed an "Extensible Table Driven Language Recognition" Utility that I wrote recently, for possible publication in your magazine. The program is written in PL/9, and runs as a FLEX utility.

I have included listings of the documentation and software that are just under 4.5 inches wide. The only way I could do this with the program listing was to use the 17CPI compressed print on a dot matrix printer. I dont know if this will reduce properly, but the source files from which these listings were made are included on the disk. Also on the disk is a compiled version.

Yours Faithfully,

Hugh Anderson.

AN EXTENSIBLE TABLE-DRIVEN LANGUAGE RECOGNITION UTILITY.

This is one of those programs that should be part of every programmers toolbox. It is specifically designed for those times when - after working on a program for two or three months - you ask yourself the question - "In what language is it written?".

Quick as a flash now - out with RECOG:

```
+++I myfile recog
```

This utility has another function - If it recognizes the source program, it is extremely likely that the source is a terrible program, as this utility searches for the seamier side of each language.

The utility is written in PL9. If you do not want to type it in, send me a SASE and a check for US$999.95, and I will return a disk complete with the source (!!).

As a first test, try it out on itself:

```
+++I recog.txt recog
```

Within four seconds the utility will recognise recog.txt as a FORTRAN program - RECOG once again shows its amazing utility.

**Ed's Note: 68 Micro Journal offers a little better deal. RECOG can be had in source and .CMD files for the price of a "Reader Service Disk" (see inset - Reader Service Disk).

DMW

- - -

```
CONSTANT
   TRUE    = -1,
   FALSE   = 0;

GLOBAL BYTE inout(16),
            eof;

BYTE table

   'FORTRAN        ','COMMON',','          ',
   'COBOL          ','ALTER ',','          ',
   'CHIP8          ',',FD 55',','          ',
   'PL1            ','DO:',','             ',
   'TRAC           ','###',','             ',
   'SNOBOL         ',',RETURN',','         ',
   'SMALLTALK      ',',StrangeCapRule',',' ',
   'Algol68        ','refrefref',','       ',
   'NEAT3          ',',000000P',','        ',
   'PROLOG         ','(is,',','            ',
   'WHIMSICAL      ',',HEX ',','           ',
   'BASIC          ',',POKE ',','          ',
   'PASCAL         ','RELEASE ',','        ',
   'C              ','for(;;)',','         ',
   'Burrough$Algol','MASK$SEARCH',','      ',
   'IBM ASSEMBLER  ','BALR ',','           ',
   'LISP           ',',))))))))))))))))','',
   'PL9            ','GBH ',','            ',
   'APL            ','''P%+0 HELL&%',',' ',
   'FORTH          ',',POLY DUP DUP',','   ',
   0:

CONSTANT
   FLEX_putchar = $CD18,
   FLEX_getchar = $CD15,
   SP           =  $20,
   CR           =  $0A,
   LF           =  $0D;

PROCEDURE getchar:
   CALL FLEX_getchar;
ENDPROC ACCA;

PROCEDURE getch:
INTEGER I:
   I = 0;
   WHILE I<15
      BEGIN
         inout( I ) = inout( I+1 );
         I          = I+1;
      END;
   inout( I ) = getchar;
   IF inout( I )=$1A THEN
      eof = TRUE;
ENDPROC;

PROCEDURE putchar( BYTE char );
   ACCA = char;
   CALL FLEX_putchar;
ENDPROC;
```

```
PROCEDURE crlf;
   putchar( CR );
   putchar( LF );
ENDPROC;

PROCEDURE print( BYTE .text );
   WHILE text
      BEGIN
         putchar( text );
         .text = .text+1;
      END;
ENDPROC;

PROCEDURE print_crlf( BYTE .text );
   print( .text );
   crlf;
ENDPROC;

PROCEDURE initialize;
INTEGER I;
   eof = FALSE;
   I   = 16;
   WHILE I
      BEGIN
         I        = I-1;
         input( I ) = SP;
      END;
ENDPROC;

PROCEDURE compare( BYTE .sample );
BYTE .comp;
   .sample = .sample+16;
   .comp   = .input;
   WHILE sample
      BEGIN
         IF sample()comp THEN
            RETURN FALSE;
         .sample = .sample+1;
         .comp   = .comp+1;
      END;
ENDPROC TRUE;

PROCEDURE lookup_table( INTEGER .language );
   language = 0;
   WHILE table( language+32 )
      BEGIN
         IF compare( .table( language+32 ) ) THEN
            RETURN TRUE;
         language = language+1;
      END;
ENDPROC FALSE;

PROCEDURE testfile;
INTEGER which_language;
   REPEAT
      getch;
      IF lookup_table( .which_language ) THEN
         BEGIN
            crlf;
            print( 'This is an awful program written in ' );
            print_crlf( .table( which_language+32 ) );
            BREAK;
         END;
   UNTIL eof;
   IF eof THEN
      BEGIN
         crlf;
         print_crlf( 'Sometimes a program does not give a desired result' );
         print_crlf( 'through programming error, or through invalid data' );
         print_crlf( 'or even through some hardware fault. This could be' );
         print_crlf( 'one of those times.' );
      END;
ENDPROC;

PROCEDURE main1
   initialize;
   testfile;
```

FLEX 2/9 DELETE UTILITY (DELETE.CMD)

DONALD E. GOULETTE
BOX 1153
FABENS, TEXAS 79838
PHONE 915-764-3607

This article describes an enhancement of the TSC FLEX 2.0 DELETE.CMD utility. This new version should work for both FLEX 2 and 9 systems (I can only vouch for FLEX 2.0).

This enhancement operates just like the old one, with one new addition. By entering +++DELETE only, you can easily delete a multitude of files without the burden of having to type in the names of all the files to be deleted.

At first thought, I was contemplating an expansion of the FLEX XOUT.CMD but found that, for myself, it was too dangerous without at least some sort of minimal prompting mechanism. The following listing is the result of what I have found to be real handy when deleting numerous files on a disk. Simply type 'DELETE<cr>' and you will receive the following:

"Delete All files (A), SOME (S), or return to FLEX (F) ?"

Enter an 'A' and DELETE will prompt for the drive number. DELETE will then prompt you to make sure. If you type 'Y' then bye-bye files. The only way to abort the mass deletion process is to type the TTYSET 'ES' (escape character) set for your system. The deletion process will stop for the NEXT file in line to be deleted. At that time, type a <cr> and you will be returned to FLEX. I don't recommend using 'A' 2:00 AM in the morning unless your really awake.

Enter a 'S' and DELETE will prompt for the drive number. DELETE will then do an individual prompt for every file on the disk. Reply either 'Y' (yes), 'N' (no), or 'F' (return to FLEX). I feel more comfortable with this mode then the 'A' mode.

The assembled listing is pretty much self documenting and should be fairly easy to make any changes that you may want. The only hazy area is the print queue handler, which wasn't to clear to me because all I did was implement the same algorithm of the old DELETE.CMD. After looking far and wide, there was no info to be had about FLEX 2.0 print queue entry points, ect. I would give just about anything for the source listing of FLEX 2.0 if and when it goes public domain. After putting up with CP/M all day at work, it sure is enjoyable to come home to FLEX.

CONCLUSION

Hope there are no major problems for the FLEX 9 version of this program. The only two areas I'm not sure of are 'QCNT' and 'QPNTR' in the listing of equates. Disemble your FLEX 9 DELETE.CMD and PEEK around, that's what I did. It's more fun that way.

```
                        NAM   DELETE.CMD
                        OPT   PAG

                *       DELETE.CMD UTILITY (ENHANCED VERSION)

                *       DONALD COULETTE
                *       BOX 1153
                *       FARENG, TEXAS 79838
                *       (915)-764-3607

                *

                *SYNTAX FOR 'DELETE.CMD':

                *       +++DELETE,TEST.BIN
                *       +++DELETE,1.TEST.TXT,0.CARD.TXT
                *       +++DELETE

                *  The first example will delete the file
                *named TEST.BIN from the working drive. The file
                *will be deleted from the first drive it is found on
                *if auto drive searching is on. The second line will
                *delete TEST.TXT from drive 1 and CARD.TXT from drive 0.
                *The third line will prompt to delete ALL or SOME
                *files.
                *  All the original DELETE.CMD restrictions are
                *retained. A file that is write or delete protected
                *or in the print queue, can't be deleted.


                *DOS EQUATES:

                *WITH THE FOLLOWING EQUATE, USE $A000 (FLEX 2.0)
                *OR $C000 (FLEX 9.0).
                *NOTE: FLEX 9.0 SHOULD WORK BUT I DON'T REALLY KNOW
                *BECAUSE I'M ONLY RUNNING FLEX 2.0.
                *    THE ONLY UNDOCUMENTED EQUATES ARE 'QCNT'
                *AND 'QPNTR' WHICH I COULD NOT FIND ANY INFO ON IN
                *THE TSC PROGRAMMER'S MANUAL. I ASSUME THEY ARE THE PRINT
                *QUEUE COUNTER ('QCNT') AND QUEUE POINTER ('QPNTR')
                *WHICH I DETERMINED BY DISASSEMBLING THE OLD
                *DELETE.CMD FILE.

A000            FLEX    EQU   $A000     START OF FLEX 2.0

                *SYSTEM EQUATES:
A718            QCNT    EQU   FLEX+$718 QUEUE COUNTER
A719            QPNTR   EQU   FLEX+$719 QUEUE POINTER
A810            TRKSEC  EQU   FLEX+$810 TRACK/SECTOR
A840            FCB     EQU   FLEX+$840 FILE CONTROL BLOCK
A020            GETFIL  EQU   FLEX+$020 GET FILE SPECS
A01E            PMSG    EQU   FLEX+$01E PRINT STRING
A024            PCRLF   EQU   FLEX+$024 CR/LF
A015            GETCHR  EQU   FLEX+$015

A03F            RPTERR  EQU   FLEX+$03F
A039            OUTDEC  EQU   FLEX+$039
A018            PUTCHR  EQU   FLEX+$018
A003            WARMS   EQU   FLEX+$003
B403            FMSCLS  EQU   FLEX+$1403
B406            FMS     EQU   FLEX+$1406

A100                    ORG   FLEX+$100 UTILITY AREA
A100 20 01      START   BRA   ST1
A102 01         VN      FCB   1         VERSION NUMBER

A103 7F A4 66   ST1     CLR   FLAG      INIT
A106 7F A4 6C           CLR   SOMFG     SOME FILES FLAG
A109 CE A8 40   ST1A    LDX   #FCB      SETUP
A10C BD AD 20           JSR   GETFIL    GET FILE SPECS
A10F 25 77              BCS   ERR1      ERROR
A111 7C A4 66           INC   FLAG      LOCKOUT
A114 A6 0C              TST   12,X      EXT PRESET?
A116 26 08              BNE   ST2       NO
A118 CE A3 88           LDX   #EXTM     NO EXT MSG
A11B BD AD 1E           JSR   PMSG
A11E 20 E9              BRA   ST1A      ANOTHER FILE
A120 86 01      ST2     LDA A #$01      OPEN FOR READ
A122 A7 00              STA A 0,X       FUNCTION
A124 BD B4 06           JSR   FMS
A127 27 03              BEQ   ST2A      OK
A129 7E A1 AF           JMP   ERR2      ERROR
A12C 86 04      ST2A    LDA A #$04      CLOSE FILE
A12E A7 00              STA A 0,X       SET FUNCTION
A130 BD B4 06           JSR   FMS
A133 26 4A              BNE   ERR0      ERROR
A135 BD A3 13           JSR   CHKQUE    GO CHECK PRINT QUE
A138 26 03              BNE   ST2B      OK
A13A 7E A1 B5           JMP   FQUE      FILE IN QUE!
A13D CE A3 4F   ST2B    LDX   #DELM     "DELETE?" MSG
A140 BD AD 24           JSR   PCRLF
A143 BD AD 1E           JSR   PMSG
A146 CE A8 42           LDX   #FCB+2    ACTIVITY BYTE
A149 BD A2 02           JSR   PRINT     PRINT FILE NAME
A14C CE A3 77           LDX   #QM       QUOTES
A14F C6 04              LDA B #$04
A151 BD A3 07           JSR   PEXT      PRINT EXT
A154 BD AD 15           JSR   GETCHR    Y/N?
A157 84 5F              AND A #$5F      MAKE UPPER CASE
A159 81 59              CMP A #'Y       YES?
A15B 26 AC              BNE   ST1A      NO
A15D CE A3 7C           LDX   #AREU     ARE YOU SURE?
A160 BD AD 1E           JSR   PMSG
A163 BD AD 15           JSR   GETCHR    Y/N?
A166 84 5F              AND A #$5F      UPPER ONLY
A168 81 59              CMP A #'Y       YES?
A16A 27 07              BEQ   DELIT     GO DELETE IT!
A16C 20 98              BRA   ST1A      NEXT FILE
A16E BD AD 1E   ST3     JSR   PMSG

A171 20 96              BRA   ST1A      NEXT FILE
A173 CE A8 40   DELIT   LDX   #FCB      LETS DELETE IT NOW!
A176 86 0C              LDA A #$0C      DELETE FUNCTION
A178 A7 00              STA A 0,X
A17A BD B4 06           JSR   FMS       DO DELETE
A17D 27 8A              BEQ   ST1A      NEXT FILE IF ANY
A17F BD AD 3F   ERR0    JSR   RPTERR    REPORT ERROR
A182 BD B4 03           JSR   FMSCLS    CLOSE FILE
A185 7E AD 03   DONE    JMP   WARMS     DONE
A188 7D A4 66   ERR1    TST   FLAG      FIRST TIME?
A188 26 F8              BNE   DONE      THATS IT!
A18D 7C A4 66           INC   FLAG      LOCKOUT
A190 FF A4 6A           STX   IDXST     SAVE
A193 BD AD 24   AGN1    JSR   PCRLF
A196 CE A3 B5           LDX   #ALLM     ALL MSG
A199 BD AD 1E           JSR   PMSG
A19C BD AD 15           JSR   GETCHR    A/S/F?
A19F 84 5F              AND A #$5F
A1A1 81 46              CMP A #'F       FLEX?
A1A3 27 E0              BEQ   DONE      YES
A1A5 81 53              CMP A #'S       SOME FILES?
A1A7 27 12              BEQ   SOMDEL     YES
A1A9 81 41              CMP A #'A       ALL FILES?
A1AB 27 11              BEQ   ALLFIL     GO TO IT
A1AD 20 D6              BRA   DONE      THATS IT
A1AF BD AD 3F   ERR2    JSR   RPTERR    REPORT ERROR
A1B2 7E A1 09           JMP   ST1A      ANOTHER
```

```
A1B5 CE A3 9E FQUE   LDX   #FILEM   FILE IN QUE
A1B8 7E A1 6E        JMP   ST3      PT IT
A1BB 7C A4 6C SQMVBC INC   SQMFG    DEL SOME FILES

                *DELETE ALL FILES HERE....

A1BE BD AD 24 ALLFIL JSR   PCRLF
A1C1 CE A4 2D        LDX   #DRVN     DRIVE # REQ
A1C4 BD AD 1E        JSR   PMSG
A1C7 BD AD 15        JSR   GETCHR    0-3
A1CA 81 30           CMP A #'0       ZERO?
A1CC 25 F0           BCS   ALLFIL    <'0'
A1CE 81 34           CMP A #$34      >'3'?
A1D0 24 EC           BCC   ALLFIL    >'3'
A1D2 36              PSH A           SAVE ASCII DRV #
A1D3 84 03           AND A #$03      STRIP ASCII
A1D5 B7 A4 69        STA A DRVN1     SAVE DRV NUMBER
A1D8 BD AD 24        JSR   PCRLF
A1DB 7D A4 6C        TST   SQMFG     SOME FILES MODE?
A1DE 26 21           BNE   ALL1      YES
A1E0 CE A3 EE        LDX   #ALMSG    ALL FILES MSG
A1E3 BD AD 1E        JSR   PMSG
A1E6 32              PUL A           DRV # IN ASCII
A1E7 BD AD 18        JSR   PUTCHR
A1EA CE A4 23        LDX   #AMSG1    PROMPT
A1ED BD A2 C5        JSR   PMSG1     PT W/O CR-LF
A1F0 BD AD 15        JSR   GETCHR    Y/N?
A1F3 36              PSH A
A1F4 BD AD 24        JSR   PCRLF
A1F7 32              PUL A
A1F8 84 5F           AND A #$5F
A1FA 81 59           CMP A #'Y       YES?
A1FC 27 03           BEQ   ALL1      ALL FILES
A1FE 7E A1 85        JMP   DONE      ABORT
A201 CE A8 40 ALL1   LDX   #FCB
A204 B6 A4 69        LDA A DRVN1     FETCH DRIVE NUMBER
A207 A7 03           STA A 3,X       SET DRV
A209 86 06           LDA A #$06      OPEN DIRECTORY
A20B A7 00           STA A 0,X       OPEN
A20D BD B4 06        JSR   FMS
A210 27 0F           BEQ   ALL2      OK
A212 CE A4 4A ERR3   LDX   #SYXM     SYNTAX MSG
A215 BD AD 1E        JSR   PMSG      PT
A218 7E A1 85        JMP   DONE
A21B BD AD 3F ERR2   JSR   RPTERR    FMS ERR REPORT
A21E 7E A1 85        JMP   DONE
A221 CE A8 40 ALL2   LDX   #FCB
A224 86 07           LDA A #$7       GET SYS INFO
A226 A7 00           STA A 0,X
A228 BD B4 06        JSR   FMS
A22B 26 EE           BNE   ERR2      FMS ERROR
A22D CE A8 40        LDX   #FCB
A230 A6 04           LDA A 4,X       FETCH 1ST CHR
A232 81 FF           CMP A #$FF      DELETED?
A234 27 EB           BEQ   ALL2      SKIP IT
A236 A6 0F           LDA A 15,X      ATTRIBUTES
A238 84 10           AND A #$10      CAT PROTECTED?
A23A 26 E5           BNE   ALL2      YES, SKIP IT
A23C A6 0F           LDA A 15,X      AGN
A23E 84 40           AND A #$40      DELETE PROTECTED?
A240 26 0F           BNE   ALL2      YES, SKIP IT
A242 A6 04           LDA A 4,X       NAME
A244 81 00           CMP A #$00      LAST ENTRY?
A246 26 03           BNE   ALL3      NOT DONE
A248 7E A1 85        JMP   DONE
A24B CE A8 40 ALL3   LDX   #FCB
A24E BD A3 13        JSR   CHKQUE    GO CHECK PRINT QUE
A251 26 09           BNE   ALL4      OK
A253 CE A3 9E        LDX   #FILEM    FILE IN QUE
A256 BD AD 1E        JSR   PMSG
A259 7E A2 21        JMP   ALL2      NEXT FILE
A25C BD AD 24 ALL4   JSR   PCRLF
A25F CE A8 42        LDX   #FCB+2    ACTIVITY BYTE
A262 BD A2 D2        JSR   PRINT     PRINT FILE NAME
A265 C6 04           LDA B #4
A267 BD A3 07        JSR   PEXT      PRINT EXT
```

```
                *A1 THIS POINT WE WILL XFER FCB TO A TEMP FCB
A26A CE A8 40        LDX   #FCB      SOURCE
A26D FF A4 6D        STX   TMPS
A270 CE A4 72        LDX   #FCB1     DEST
A273 FF A4 6F        STX   TMPD
A276 C6 10           LDA B #16       XFER WIDTH
A278 FE A4 6D XFER   LDX   TMPS      SOURCE
A27B A6 00           LDA A 0,X       FETCH SOURCE BYTE
A27D 08              INX
A27E FF A4 6D        STX   TMPS      UPDATE
A281 FE A4 6F        LDX   TMPD      DEST
A284 A7 00           STA A 0,X       SAVE
A286 08              INX
A287 FF A4 6F        STX   TMPD      UPDATE
A28A 5A              DEC B
A28B 26 EB           BNE   XFER      MORE TO XFER
A28D 7D A4 6C        TST   SQMFG     DEL SOME MODE?
A290 27 15           BEQ   SQM1      NO!
A292 CE A3 38        LDX   #DELMS    DELETE MSG
A295 BD A2 C5        JSR   PMSG1
A298 BD AD 15        JSR   GETCHR    Y/N?
A29B 84 5F           AND A #$5F      UPPER CASE
A29D 81 59           CMP A #'Y       YES?
A29F 27 06           BEQ   SQM1      YES!
A2A1 81 46           CMP A #'F       FLEX?
A2A3 27 1A           BEQ   ALL6A     YES
A2A5 26 15           BNE   ALL6      SKIP IT
A2A7 CE A4 72 SQM1   LDX   #FCB1     LETS DELETE IT NOW
A2AA 86 0C           LDA A #$0C      DELETE FUNCTION
A2AC A7 00           STA A 0,X
A2AE BD B4 06        JSR   FMS
A2B1 27 03           BEQ   ALL5      OK
A2B3 7E A1 7F ALL4A  JMP   ERR0      REPORT ERR
A2B6 CE A4 5B ALL5   LDX   #FDM
A2B9 BD A2 C5        JSR   PMSG1
A2BC 7E A2 21 ALL6   JMP   ALL2      NEXT FILE
A2BF 7E A1 85 ALL6A  JMP   DONE      ABORT

                *DELETE SOME FILES HERE:

A2C2 7E A1 85 SQMFIL JMP   DONE
```

```
                *<<<<<<<<<<<<<<<< PMSG1 >>>>>>>>>>>>>>>>>>>
                *PMSG W/O CRLF

A2C5 A6 00    PMSG1  LDA A 0,X       FETCH CHR
A2C7 81 04           CMP A #4        EOT?
A2C9 27 06           BEQ   PMSG2     DONE
A2CB BD A0 18        JSR   PUTCHR    PR IT
A2CE 08              INX
A2CF 20 F4           BRA   PMSG1     ANOTHER CHR
A2D1 39       PMSG2  RTS

                *<<<<<<<<<<<<<<<< PRINT >>>>>>>>>>>>>>>>>>>
                *PRINT FILE NAME.EXT HERE....

A2D2 6F 00    PRINT  CLR   0,X       CLR ACTIVITY STATUS
A2D4 5F              CLR B           NO LEAD ZEROS
A2D5 BD AD 39        JSR   OUTDEC    PRINT DRV #
A2D8 CE A8 44        LDX   #FCB+4    FILE NAME
A2DB 86 2E           LDA A #'.       DECIMAL POINT
A2DD BD AD 18        JSR   PUTCHR
A2E0 C6 08           LDA B #8        8 CHRS MAX
A2E2 86 09           LDA A #9
A2E4 B7 A4 71        STA A CNT
A2E7 A6 00    FQ0    LDA A 0,X       FETCH FILE NAME CHR
A2E9 27 06           BEQ   FQ1       NOP!
A2EB BD AD 18        JSR   PUTCHR    PRINT CHR
A2EE 7A A4 71        DEC   CNT
A2F1 08       FQ1    INX             NEXT FILE NAME CHR
A2F2 5A              DEC B           1 LESS CHR
A2F3 26 F2           BNE   FQ0       CONTINUE NAME PRINT
A2F5 F6 A4 71        LDA B CNT       SPACE CNTR
```

FEATURES THE POWERFUL, THIRD GENERATION, MOTOROLA 6809 PROCESSOR!

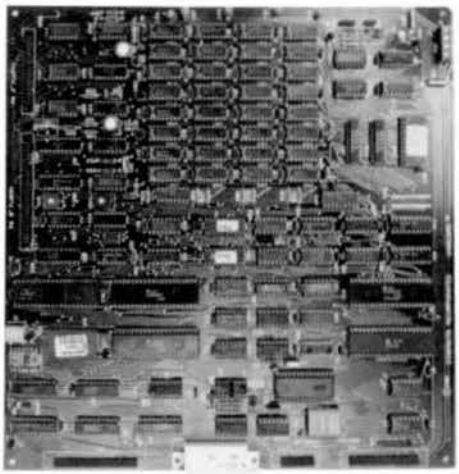
THE 6809 "UNIBOARD"™

SINGLE BOARD COMPUTER KIT

PERFECT FOR COLLEGES, OEM'S, INDUSTRIAL AND SCIENTIFIC USES!

64K RAM! DOUBLE DENSITY FLOPPY DISK CONTROLLER!

SPECIAL CLOSE-OUT *Lower Price!*



BLANK PC BOARD

$49.95

WITH PAL'S, AND TWO EPROMS.

FOR 5-1/4 OR 8 INCH SOURCE DISKETTE ADD $10.

SPECIAL
CLOSE-OUT

PRICE CUT!!

THE COMPACTA UNIBOARD™: Through special arrangement with COMPACTA INC., we are proud to have been selected the exclusive U.S. Mfg. of their new 6809 UNIBOARD™ COMPUTER KIT. Many software professionals feel that the 6809 features probably the most powerful instruction set available today on ANY 8 bit micro. Now, at last, all of that immense computing power is available at a truly unbelievably low price.

★ Includes Patches & Fixes on Disk as published in March 86 68' Micro Journal

FEATURES:

- ★ 64K RAM using 4116 RAMS.
- ★ 6809E Motorola CPU.
- ★ Double Density Floppy Disk Controller for either 5-1/4 or 8 inch drives. Uses WD1793.
- ★ On board 80 x 24 video for a low cost console. Uses 2716 Char. Gen. Programmable Formats. Uses 6845 CRT Controller.
- ★ ASCII keyboard parallel input interface. (6522)
- ★ Serial I/O (6551) for RS232C or 20 MA loop.
- ★ Centronics compatible parallel printer interface. (6522)
- ★ Buss expansion interface with DMA channel. (6844)
- ★ Dual timer for real time clock application.
- ★ Powerful on board system monitor (2732). Features commands such as Go To, Alter, Fill, Move, Display, or Test Memory. Also Read and Write Sectors. Boot Normal, Unknown, and General Flex™.

YOUR CHOICE OF POPULAR DISK OPERATING SYSTEMS:

FLEX™ from TSC	$99
OS9™ from Microware	$199

Specify 5-1/4 or 8 Inch

PC BOARD IS DOUBLE SIDED, PLATED THRU SOLDER MASKED, 11 x 11-1/2 IN.

ALL SALES ARE MADE SUBJECT TO THE TERMS OF OUR 90 DAY LIMITED WARRANTY. A FREE COPY IS AVAILABLE UPON REQUEST.

Digital Research Computers
(OF TEXAS)

P.O. BOX 381450 DUNCANVILLE, TX. 75138 (214) 225-2309

TERMS: Shipments will be made approximately 3 to 6 weeks after we receive your order. VISA, MC, cash accepted. Add $4.00 shipping. USA AND CANADA ONLY

```
A2F8 86 20    FQ2     LDA A  #$20     SPACE
A2FA BD AD 1B         JSR    PUTCHR
A2FD 5A               DEC B
A2FE 26 FB            BNE    FQ2      MORE
A300 C6 03            LDA B  #$03     3 CHR PRINT
A302 86 2E            LDA A  #'.      DECIMAL POINT
A304 BD AD 1B         JSR    PUTCHR   PT IT
A307 A6 00    PEXT    LDA A  0,X      GET EXT CHR
A309 27 07            BEQ    PEXT1    MAP, RET
A30B BD AD 1B         JSR    PUTCHR   PT IT
A30E 08               INX             NEXT EXT CHR
A30F 5A               DEC B           CHR CNT
A310 26 F5            BNE    PEXT     MORE TO PRINT
A312 39       PEXT1   RTS

              *<<<<<<<<<<<<<<<<<< CHKQUE >>>>>>>>>>>>>>>>>>
              *CHECK FOR FILE IN QUEUE.....

A313 B6 A7 1B CHKQUE  LDA A  QCNT     QUEUE CNTR
A316 B7 A4 67         STA A  NOFILS   SAVE IT
A319 27 31            BEQ    DONE1    NO FILES
A31B A6 03            LDA A  3,X      DRIVE #
A31D B7 A4 68         STA A  DRVN     SAVE DRV #
A320 A6 11            LDA A  17,X     TRACK
A322 E6 12            LDA B  18,X     SECTOR
A324 FE A7 19         LDX    QPNTR    PRINT Q POINTER
A327 A1 01    QUE1    CMP A  1,X      TRACK MATCH?
A329 26 0E            BNE    CHK      NO MATCH
A32B E1 02            CMP B  2,X      SECTOR MATCH?
A32D 26 0A            BNE    CHK      NO MATCH
A32F 36               PSH A
A330 B6 A4 68         LDA A  DRVN     GET DRV #
A333 A1 00            CMP A  0,X      SAME DRV?
A335 32               PUL A
A336 26 01            BNE    CHK      NO MATCH
A338 39               RTS
A339 7A A4 67 CHK     DEC    NOFILS   DEC #Q FILE CNT
A33C 27 0E            BEQ    DONE1    THATS ALL FOLKS!
A33E 08               INX
A33F 08               INX
A340 08               INX             ON TO NEXT Q FILE
A341 08               INX
A342 8C A8 40         CPX    #FCB     REACHED LIMIT?
A345 26 03            BNE    CHK1     NO, TEST MORE IN Q
A347 CE A8 10         LDX    #TRKSEC  START OF Q MAP
A34A 20 08    CHK1    BRA    QUE1     GO TST
A34C B6 01    DONE1   LDA A  #$01     DONE
A34E 39               RTS

              *SCREEN MESSAGES:

A34F 44       DLLM    FCC    'Delete '
A357 04               FCB    4
A358 20       DELM1   FCC    '  DELETE this file (Y/N/F) ? '
A376 04               FCB    4
A377 22       QM      FCC    '" ? '
A37B 04               FCB    4
A37C 41       AREM    FCC    'Are you sure? '
A38A 04               FCB    4
A38B 45       EXTM    FCC    'Extension required'
A39D 04               FCB    4
A39E 46       FILEM   FCC    'File is in print queue'
A3B4 04               FCB    4
A3B5 44       ALLM    FCC    'Delete ALL files (A), SOME (S), '
A3D5 6F               FCC    'or return to FLEX (F) ? '
A3ED 04               FCB    4
A3EE 41       AMSG    FCC    'Are you sure you want to delete '
A40E 41               FCC    'ALL FILES on drive #'
A422 04               FCB    4
A423 20       AMSG1   FCC    ' (Y/N) ? '
A42C 04               FCB    4
A42D 44       DRVM    FCC    'DRIVE NUMBER PLEASE (0-3) ? '
A449 04               FCB    4
A44A 53       SYXM    FCC    'SYNTAX ERROR !!!'
A45A 04               FCB    4
A45B 20       FDM     FCC    '  Deleted'
A465 04               FCB    4

              *STORAGE LOCATIONS:

A466          FLAG    RMB    1
A467          NOFILS  RMB    1        NUMBER OF Q FILES
A468          DRVN    RMB    1        DRV#
A469          DRVN1   RMB    1        *
A46A          IDIST   RMB    2        IDX REG STORAGE
A46C          SOMFQ   RMB    .1       SOME FILES FLAG

A46D          TMPS    RMB    2
A46F          TMPD    RMB    2
A471          CNT     RMB    1        CHR CNTR
A472          FCB1    RMB    320      TRANSIT FCB

A8B2          ZEND    EQU    *

                      END    START

NO ERROR(S) DETECTED

   SYMBOL TABLE:

AGN1  A193  ALL1  A201  ALL2  A221  ALL3  A24B  ALL4  A25C
ALL4A A283  ALL5  A2B6  ALL6  A2BC  ALL6A A2BF  ALLFIL A18E
ALLM  A385  AMSG  A3EE  AMSG1 A423  AREM  A37C  CHK   A339
CHK1  A34A  CHKQUE A313  CNT   A471  DELIT  A173  DELM  A34F
DELM1 A358  DONE  A185  DONE1 A34C  DRVM  A42D  DRVN  A468
DRVN1 A469  ERR0  A17F  ERR1  A188  ERR2  A1AF  ERR3  A212
ERR7  A21B  EXTM  A38B  FCB   A840  FCB1  A472  FDM   A45B
FILEM A39E  FLAG  A466  FLEX  A000  FMS   B406  FMSCLS B403
FQ0   A2E7  FQ1   A2F1  FQ2   A2FB  FQUE  A185  GETCHR AD15
GETFIL AD20 IDIST A46A  NOFILS A467  OUTBEE AD39  PCRLF AD24
PEXT  A307  PEXT1 A312  PMSG  AD1E  PMSG1 A2C5  PMSG2 A2D1
PRINT A2D2  PUTCHR AD1B  QCNT  A71B  QM    A377  QPNTR A719
QUE1  A327  RPTERR A03F  SOM1  A2A7  SOMFQ A46C  SOMFIL A2C2
SOMVEC A1BB  ST1   A103  ST1A  A109  ST2   A120  ST2A  A12C
ST2B  A130  ST3   A16E  START A100  SYXM  A44A  TMPD  A46F
TMPS  A46D  TRKSEC A810  VN    A102  WARMS AD03  XFER  A27B
ZEND  A8B2
```

Classified Advertising

Winchester 10 Megabyte Drives

Two (2) 10 Megabyte Hard-Disk Winchester Drives. Working - were removed for upgrade to larger drives.

1 - RMS Model #509 $275.00
1 - Seagate Model #412 $275.00

(615) 842-4600 Tom 9-5 EST.

LSI 68008 CPU Card with Digital Research CPM/68K $350.

Tano Qutpost II, 56K, 2 5"DSDD Drives, FLEX, MUMPS $895.

MICROKEY $4500 Single Board Computer, Target 128K RAM, FLEX, FORTH, with optional 6502 CPU & ROMS as advertised on p. 51 DEC. 84 68' Micro Journal. $2300.

1-PT-69 complete with Dual 5" DSDD Disk System and Controller, includes FLEX DOS. $745.

TELETYPE Model 43 PRINTER - with serial (RS232) interface and full ASCII keyboard. LIKE NEW - new cost $1295.00 - ONLY $559.00 ready to run.

S/09 with Motorola 128K RAM, 1-MPS2, 1-Parallel Port, MP-09 CPU Card $1590.

1-CDS1 20 Meg Hard Disk System with Controller $1800.

Call Tom (615) 842-4600 M-F 9 A.M. to 5 P.M. E.S.T.

***

69/K, 40K RAM, DUAL 5" DS DRIVES, MPS2, CLOCK-TIMER-PARALLEL PORT, 8212 TERMINAL, FLEX SOFTWARE $1700 or B/O. CALL BOB (412) 793-6558 EVENINGS.

***

FOR SALE: Gimix 6809, OS9 Level II, 128K; Clock; 2 serial ports; 2 Tandon DSDD 80's; Televideo 950 Terminal; IDS 460 Printer. Software: Flex, Assembler, TSC Utilities, Diagnostics, Stylo, Dynacalc, Sort-Merge, XBasic, eForth, OS9, and more. Cost new over $7000. Asking 2900.00. Call 315-475-1548, 9-5 EST M-F.

***

SWTPC CT-8212 Terminal in original shipping container. Call evenings 603-329-5811.

***

ATTENTION SWTP USERS - S/09 Upgradeable to S+, 64K Memory, 8" Floppy, 10Mb Winchester, 6 Serial Ports, 1 Parallel Port, Software, Manuals, Qume Terminal. Make Offer! Call (314) 469-3100.

WINDRUSH MICRO SYSTEMS

UPROM II

PROGRAMS and VERIFIES: I2758, T2508, I2716, T2516, I2732/2732A, MCM68764/6, I2764/2764A, T2564, I27128/27128A, and I27256. I=Intel, T=Texas, M=Motorola.

NO PERSONALITY MODULES REQUIRED!

TRI-VOLT EPROMS ARE NOT SUPPORTED

INTEL's intelligent programming (tm) implemented for Intel 2764, 27128 and 27256 devices. Intelligent programming reduces the average programming time of a 2764 from 7 minutes to 1 minute 15 seconds (under FLEX) with greatly improved reliability.

Fully enclosed pod with 5' of flat ribbon cable for connection to the host computer MC6821 PIA interface board.

MC6809 software for FLEX and OS9 (Level 1 or 2, Version 1.2).

BINARY 9I5X FILE offset Loader supplied with FLEX, MDOS and OS9.

Menu driven software provides the following facilities:

a. FILL ..... a selected area of the buffer with a HEX char.
b. MOVE ..... blocks of data.
c. DUMP ..... the buffer in HEX and ASCII.
d. FIND ..... a string of bytes in the buffer.
e. EXAMINE/CHANGE . the contents of the buffer.
f. CRC ..... checksum a selected area of the buffer.
g. COPY ..... a selected area of an EPROM into the buffer.
h. VERIFY ..... a selected area of an EPROM against the buffer.
i. PROGRAM ..... a selected area of an EPROM with data in the buffer.
j. SELECT ..... a new EPROM type (return to types menu).
k. ENTER ..... the system monitor.
l. RETURN ..... to the operating system.
m. EXECUTE ..... any OS9 utility (only in FLEX and OS9 versions).

FLEX AND OS9 VERSIONS AVAILABLE FROM GIMIX. SSB/MDOS CONTACT US DIRECT.

PL/9

- Friendly inter-active environment where you have INSTANT access to the Editor, the Compiler, and the Trace-debugger, which, amongst other things, can single step the program a SOURCE line at a time. You also have direct access to any FLEX utility and your system monitor.
- 375+ page manual organized as a tutorial with plenty of examples.
- Fast SINGLE PASS compiler produces 8K of COMPACT and FAST 6809 machine code output per minute with no run-time overheads or license fees.
- Fully compatible with TSC text editor format disk files.
- Signed and unsigned BYTES and INTEGERS, 32-bit floating point REALs.
- Vectors (single dimension arrays) and pointers are supported.
- Mathematical expressions: (+), (-), (=), (/), modulus (\), negation (-)
- Expression evaluators: (=), (<>), (<), (>), (>=), (<=)
- Bit operators: (AND), (OR), (EOR/XOR), (NOT), (SHIFT), (SWAP)
- Logical Operators: (.AND), (.OR), (.EOR/XOR)
- Control statements: IF..THEN..ELSE, IF..CASE1..CASE2..ELSE, BEGIN..END, WHILE.., REPEAT..UNTIL, REPEAT..FOREVER, CALL, JUMP, RETURN, BREAK, GOTO.
- Direct access to (ACCA), (ACCB), (ACCD), (XREG), (CCR) and (STACK).
- FULLY supports the MC6809 RESET, NMI, FIRQ, IRQ, SWI, SWI2, and SWI3 vectors. Writing a self-starting (true power-up) program that uses ANY, or ALL, of the MC6809 interrupts is an absolute snap!
- Machine code may be embedded in the program via the 'GEN' statement. This enables you to code critical routines in assembly language and embed them in the PL/9 program (see 'MACE' for details).
- Procedures may be passed and may return variables. This makes them functions which behave as though they were an integral part of PL/9.
- Several fully documented library procedure modules are supplied: IOSUBS, BITIO, HARDIO, HEXIO, FLEXIO, SCIPACK, STRSUBS, BASTRING, and REALCOM.

'... THIS IS THE MOST EFFICIENT COMPILER I HAVE FOUND TO DATE.'

Quoted from Ron Anderson's FLEX User Notes column in '68. Need we say more?

MACE/XMACE/ASM05

All of these products feature a highly productive environment where the editor and the assembler reside in memory together. Gone are the days of tedious disk load and save operations while you are debugging your code.

- Friendly inter-active environment where you have instant access to the Editor and the Assembler, FLEX utilities and your system monitor.
- MACE can also produce ASMPROCS (GEN statements) for PL/9 with the assembly language source passed to the output as comments.
- XMACE is a cross assembler for the 6800/1/2/3/8 and supports the extended mnemonics of the 6303.
- ASM05 is a cross assembler for the 6805.

D-BUG

LOOKING for a single step tracer and mini in-line disassembler that is easy to use?? Look no further, you have found it. This Package is ideal for those small assembly language program debugging sessions. D-BUG occupies less than 2K (including its stack and variables) and may be loaded anywhere in memory. All you do is LOAD it, AIM it and GO! (80 col VDUs only).

McCOSH 'C'

This is as complete a 'C' compiler as you will find on any operating system for the 6809. It is completely compatible with UNIX VII and only lacks 'bit-fields' (which are of little practical use in an 8-bit world!).

- Produces very efficient assembly language source output with the 'C' source optionally interleaved as comments.
- Built-in optimizer will shorten object code by about 11%.
- Supports interleaved assembly language programs.
- INCLUDES its own assembler. The TSC relocating assembler is only required if you want to generate your own libraries.
- The pre-processor, compiler, optimizer, assembler and loader all run independently or under the 'CC' executive. 'CC' makes compiling a program to executable object as simple as typing in 'CC,HELLO.C <RETURN>'.

IEEE-488

- SUPPORTS ALL PRINCIPAL MODES OF THE IEEE-488 (1975/8) BUS SPECIFICATION:
  - Talker - Serial Poll - Single or Dual Primary Address
  - Listener - Parallel Poll - Secondary Address
  - System Controller - Group Trigger - Talk only ... Listen only
- Fully documented with a complete reprint of the KILOBAUD article on the IEEE bus and the Motorola publication 'Getting aboard the IEEE Bus'.
- Low level assembly language drivers suitable for 6800, 6801, 6802, 6803, 6808 and 6809 are supplied in the form of listings. A complete back to back test program is also supplied in the form of a listing. These drivers have been extensively tested and are GUARANTEED to work.
- Single S-100 board (4, 8 or 16 addresses per port), fully socketed, gold plated bus connectors and IEEE interface cable assembly.

PRICES

D-BUG	(6809 FLEX only)	$ 75.00
MACE	(6809 FLEX only)	$ 75.00
XMACE	(6809 FLEX only)	$ 98.00
ASM05	(6809 FLEX only)	$ 98.00
PL/9	(6809 FLEX only)	$198.00
'C'	(6809 FLEX only)	$295.00
IEEE-488	with IEEE-488 cable assembly	$298.00
UPROM-II/U	with one version of software (no cable or interface)	$395.00
UPROM-II/C	as above but complete with cable and S-100 interface	$545.00
CABLE	5' twist-n-flat 50 way cable with IDC connectors	$ 35.00
S-100 INF	S-100 interface for UPROM-II	$130.00
EXOR INF	Motorola EXORbus (EXORciser) interface for UPROM-II	$195.00
UPROM SFT	Software drivers for 2nd operating system. Specify FLEX or OS9 AND disk size!	$ 35.00
UPROM SRC	Assembly language source (contact us direct)	

ALL PRICES INCLUDE AIR MAIL POSTAGE

Terms: CWO. Payment by Int'l Money Order. VISA or MASTER-CARD also accepted.

WORSTEAD LABORATORIES, NORTH WALSHAM, NORFOLK, ENGLAND. NR28 9SA.

TEL: 44 (692) 404086
TLX: 975548 WMICRO G

WE STOCK THE FOLLOWING COMPANIES PRODUCTS: GIMIX, SSB, FHL, MICROWARE, TSC, LUCIDATA, LLOYD I/O, & ALFORD & ASSOCIATES.

FLEX (tm) is a trademark of Technical Systems Consultants, OS-9 (tm) is a tradeamrk of Microware Systems Corporation, MDOS (tm) and EXORciser (tm) are trademarks of Motorola Incorporated.

68' MICRO JOURNAL

Disk-1 Fileasrt, Minicat, Minicopy, Minifms, **Lifetime, **Poetry, **Foodlist, **Diet.

Disk-2 Diskedit w/ inst.& fixes, Prime, *Prmod, **Snoopy, **Football, **Hexpawn,**Lifetime

Disk-3 Cbug09, Sec1, Sec2, Find, Table2, Intext, Disk-exp, *Diskasve.

Disk-4 Mailing Program, *Finddat, *Change, *Testdisk.

DISK-5 *DISKFIX 1, *DISKFIX 2, **LETTER, **LOVESIGN, **BLACKJAK, **BOWLING.

Disk-6 **Purchase Order, Index (Disk file indx)

Disk-7 Linking Loader, Rload, Harkness

Disk-8 Crtest, Lanpher (May 82)

Disk-9 Datecopy, Diskfix9 (AuR 82)

Disk-10 Home AccountinR (July 82)

Disk-11 Dissembler (June 84)

Disk-12 Modem68 (May 84)

Disk-13 *Initmf68, Testmf68, *Cleanup, *DskeliRn, Help, Date.Txt

Disk-14 *Init, *Test, *Terminal, *Find, *Diskedit, Init.Lib

Disk-15 Modem9 + Updates (Dec. 84 Gilchrist) to Modem9 (April 84 Comm)

Disk-16 Copy.Txt, Copy.Doc, Cat.Txt, Cat.Doc

Disk-17 Match Utility, RATBAS, A Basic Preprocessor

Disk-18 Parse.Mod, Size.Cmd (Sept. 85 ArmstronR), CMDCODE, CMD.Txt (Sept. 85 Spray)

Disk-19 Clock, Date, Copy, Cat, PDEL.Asm & Doc., Errors.Sys, Do, Log.Asm & Doc.

Disk-20 UNIX Like Tools (July & Sept. 85 Taylor & Gilchrist). Dragon.C, Grep.C, LS.C, FDUMP.C

Disk-21 Utilities & Games - Date, Life, Madness, Touch, Goblin, Starshot, & 15 more.

Disk-22 Read CPM & Non-FLEX Disks. Fraser May 1984.

Disk-23 ISAM, indexed sequential file accessing methods. Gondon Nov. 1985.

Disk-24 68' Micro Journal Index of Articles & Bit Bucket Items from 1979 - 1985, John Current.

NOTE:

This is a reader service ONLY! No Warranty is offered or implied, they are as received by '68' Micro Journal, and are for reader convenience ONLY (some MAY include fixes or patches). Also 6800 and 6809 programs are mixed, as each is fairly simple (mostly) to convert to the other.

PRICE: 8" Disk **$14.95** - 5" Disk **$12.95**

68' MICRO JOURNAL
POB 794
Hixson, TN 37343
615-842-4600

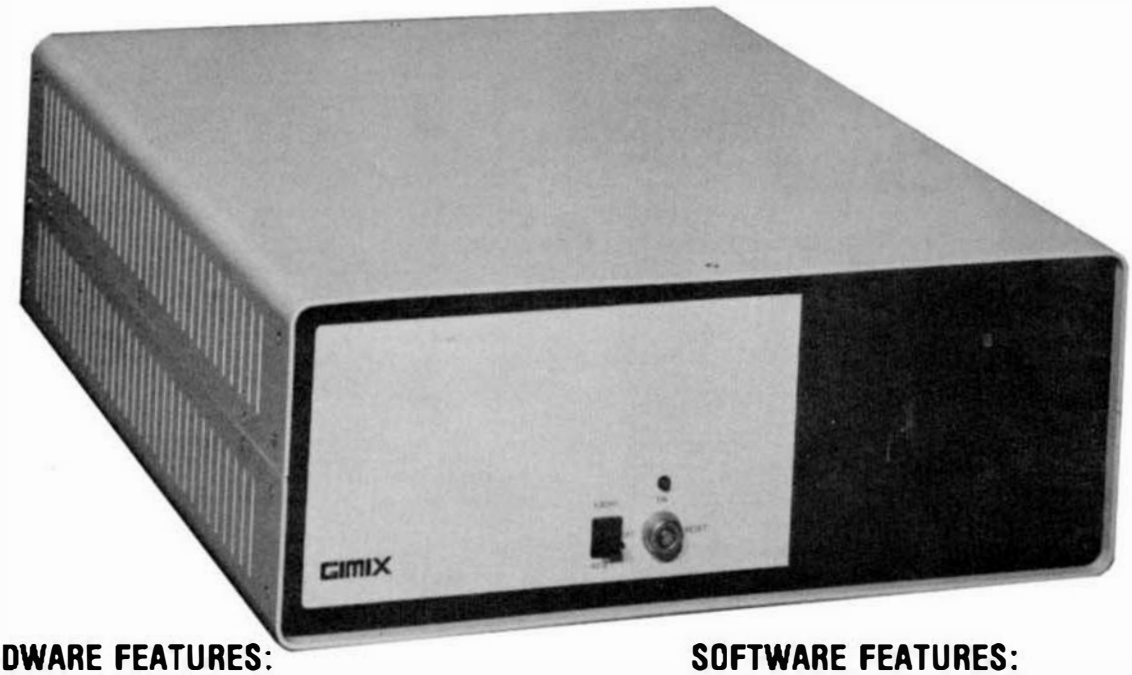
* Indicates 6800
** Indicates BASIC SWTPC or TSC
6809 no Indicator.

MASTER CARD - VISA Accepted
Foreign -- add **10%** for Surface
or **20%** for Air!!

Telex 5106006630

GMX 68020 DEVELOPMENT SYSTEM

A Multi-user, Multi-tasking software development system for use with all 68000 family processors.



HARDWARE FEATURES:

- The GMX-020 CPU board has: the MC68020 32-bit processor, a 4K Byte no wait-state instruction cache, high-speed MMU, and a full-featured hardware time of day clock/calendar with battery back-up. It also provides for an optional 68881 floating point co-processor.
- 1 Megabyte of high speed static RAM.
- Intelligent Serial and Parallel I/O Processor boards significantly reduce system overhead by handling routine I/O functions. This frees up the host CPU for running user programs. The result is a speed up of system performance and allows all terminals to run at up to 19.2K baud.
- The system hardware will support up to 39 terminals.
- Powered by a constant voltage ferro-resonant power supply that insures proper system operation under adverse AC power input conditions.
- OMA hard disk interface and DMA double density floppy disk controller are used for data transfers at full bus speed. The DMA hard disk drive controller provides automatic 22-bit burst data error detection and 11-bit burst error correction.
- A selection of hard disk drives with capacities from 19 to 85 Megabytes, removeable pack hard disk drives, streaming tape drives, and floppy disk drives is available.

SOFTWARE FEATURES:

The UniFLEX VM Operating System is a demand-paged, virtual memory operating system written in 68020 Assembler code for compactness and efficiency. Any UniFLEX system will run faster than a comparable system written in a higher level language. This is important in such areas as context switching, disk I/O, and system call handling. Other features include:

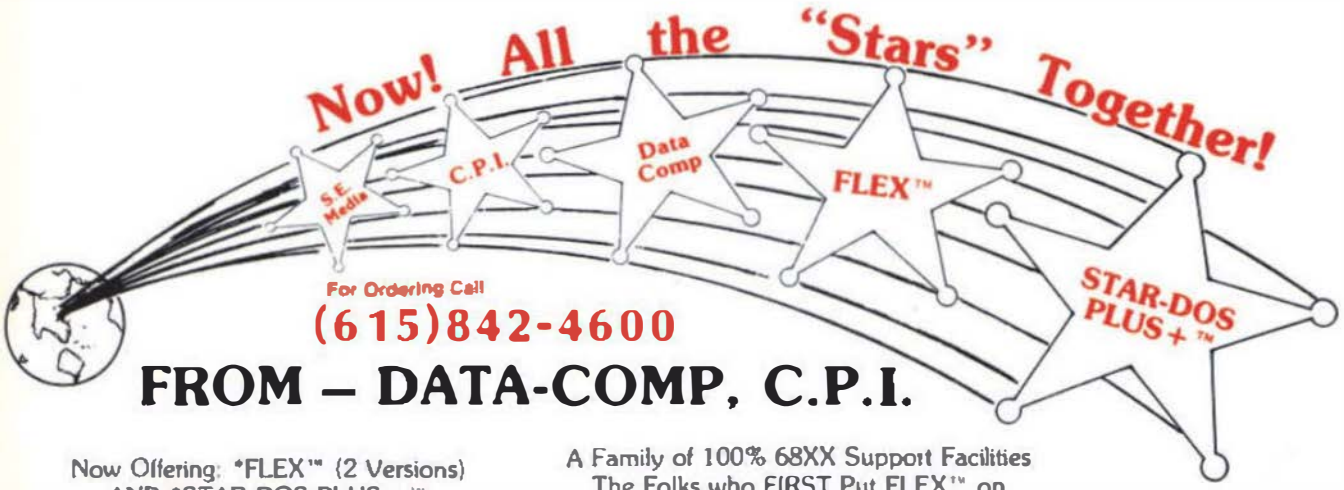
- compact, efficient Kernel and modules allows handling more users more effectively than UNIX systems, using much less disk space.
- UNIX system V compatibility at the C source code level.
- C Compiler optimized in 68020 code (optional).
- Record locking for shared files.
- Users can share programs in memory.
- Modeled after UNIX systems, with similar commands.
- System accounting facilities.
- Sequential and random file access.
- Maximum record size limited only by the disk size.
- Multiple Level Directories.
- Up to 4 Megabytes of Virtual Memory per user.
- Optional Languages available are: C, BASIC, COBOL, FORTRAN, LISP, PROLOG, SCULPTOR, and ASSEMBLER. In development are ADA, PASCAL, and FORTH.

Included with the UniFLEX Operating System are a Utilities package, editor, relocating assembler, linking loader, and printer spooler. Options inlcude a fast floating point package, library generator, and a sort-merge package.

The GMX version of the MOTOROLA 020 BUG is included with the system.

GIMIX, Inc., a Chicago based microcomputer company established in 1975, has produced state of the art microcomputer systems based on Motorola 6800 and 6809 microprocessors. GIMIX systems are in use in Industry,, Hospitals, Universities, Research Organiations, and by Software Developers. GIMIX was awarded the prestigious President's "E" Certificate for Exports in 1984.

GIMIX INC. 1337 WEST 37th PLACE • CHICAGO, ILLINOIS 60609 • (312) 927-5510 • TWX910-221-4055



For Ordering Call
(615)842-4600

FROM – DATA-COMP, C.P.I.

Now Offering: *FLEX™ (2 Versions)
AND *STAR-DOS PLUS+™

A Family of 100% 68XX Support Facilities
The Folks who FIRST Put FLEX™ on
The CoCo

FLEX-CoCo Sr.
with TSC Editor
TSC Assembler
Complete with Manuals
Reg. $250.00 **Only $79.95**

STAR-DOS PLUS+
- Functions Same as FLEX
- Reads - writes FLEX Disks
- Run FLEX Programs
- Just type: Run "STAR-DOS"
- Over 300 utilities & programs to choose from.

$34.95

FLEX-CoCo Jr.
without TSC Editor & Assembler
$49.95

PLUS

ALL VERSIONS OF FLEX & STAR-DOS• INCLUDE

+ Read-Write-Dir RS Disk
+ Run RS Basic from Both
+ More Free Utilities
+ External Terminal Program
+ Test Disk Program
+ Disk Examine & Repair Program
+ Memory Examine Program
+ Many Many More!!!

TSC Editor
Reg $50.00
NOW $35.00

TSC Assembler
Reg $50.00
NOW $35.00

CoCo Disk Drive Systems

2 THINLINE DOUBLE SIDED DOUBLE DENSITY DISK DRIVES SYSTEM WITH POWER SUPPLY,CABINET,DISK DRIVE CABLE,J&M NEW DISK CONTROLLER JFD-CP WITH J-DOS,RS-DOS OPERATING SYSTEMS. $469.95

* Specify What CONTROLLER You Want J&M, or RADIO SHACK

THINLINE DOUBLE SIDED DOUBLE DENSITY 40 TRACKS $129.95

Verbatim Diskettes

Single Sided Double Density	$ 24.00
Double Sided Double Density	$ 24.00

Controllers

J&M JFD-CP WITH J-DOS	$139.95
WITH J-DOS, RS-DOS	$159.95
RADIO SHACK 1.1	$134.95
RADIO SHACK Disk CONTROLLER 1.1	$134.95

Disk Drive Cables

Cable for One Drive	$ 19.95
Cable for Two Drives	$ 24.95

MISC

64K UPGRADE FOR C,D,E,F, AND COCO II	$ 29.95
RADIO SHACK BASIC 1.2	$ 24.95
RADIO SHACK DISK BASIC 1.1	$ 24.95
DISK DRIVE CABINET FOR A SINGLE DRIVE	$ 49.95
DISK DRIVE CABINET FOR TWO THINLINE DRIVES	$ 69.95

PRINTERS

EPSON LX-80	$289.95
EPSON MX-70	$325.95
EPSON MX-100	$495.95

ACCESSORIES FOR EPSON

8148 2K SERIAL BOARD	$ 89.95
8149 32K EXPAND TO 128K	$169.95
EPSON MX-RX-80 RIBBONS	$ 7.95
EPSON LX-80 RIBBONS	$ 5.95
TRACTOR UNITS FOR LX-80	$ 39.95

CABLES & OTHER INTERFACES CALL FOR PRICING

DATA-COMP
5900 Cassandra Smith Rd.
Hixson, TN 37343

SHIPPING
USA ADD 2%
FOREIGN ADD 5%
MIN. $2.50

(615)842-4600
For Ordering
Telex 5106006630

Introducing

S-50 BUS/68XX

Board and/or Computer

Terminals-CRTs-Printers

Disk Drives-etc.

REPAIRS

NOW AVAILABLE TO ALL S50/68XX USERS

The Data-Comp Division of CPI is proud to announce the availability of their service department facilities to 'ALL' S50 Bus and 68XX users. Including all brands, SWTPC - GIMIX - SSB - HELIX and others, including the single board computers. * Please note that kit-built components are a special case, and will be handled on an individual basis, if accepted.

1. If you require service, the first thing you need to do is call the number below and describe your problem and confirm a Data-Comp service & shipping number! This is very important, Data-Comp will not accept or repair items not displaying this number! Also we cannot advise or help you troubleshoot on the telephone, we can give you a shipping number, but NO advice! Sorry!

2. All service shipments must include both a minimum $40.00 estimate/repair charge and pre-paid return shipping charges (should be same amount you pay to ship to Data-Comp).

3. If you desire a telephone estimate after your repair item is received, include an additional $5.00 to cover long distance charges. Otherwise an estimate will be mailed to you, if you requested an estimate. Estimates must be requested. Mailed estimates slow down the process considerably. However, if repairs are not desired, after the estimate is given, the $40.00 shall constitute the estimate charge, and the item(s) will be returned unrepaired providing sufficient return shipping charges were included with item to be serviced. Please note that estimates are given in dollar amounts only.

4. Data-Comp service is the oldest and most experienced general S50/68XX service department in the world. We have over $100,000.00 in parts in stock. We have the most complete set of service documents for the various S50/68XX systems of anyone - YET, WE DO NOT HAVE EVERYTHING! But we sure have more than anyone else. We repair about 90% of all items we receive. Call for additional information or shipping instructions.

This

Not Th

MR. MICKEY FERGUSON
P. O. BOX 87
HIXSON SPRINGS TN 37082

DATA-COMP
5900 Cassandra Smith Rd.
Hixson, TN 37343

(615)842-4607
Telex 5106006630